Modeling the Fault Tolerance Consequences of Deduplication

Eric W. D. Rozier and William H. Sanders
University of Illinois
Coordinated Science Laboratory
1308 W. Main Street, Urbana, IL 61801
{erozier2,whs}@illinois.edu

Pin Zhou, Nagapramod Mandagere,
Sandeep M Uttamchandani, and Mark L. Yakushev
IBM Almaden Research Center
650 Harry Road, San Jose, California 95120
{pinzhou, nmandage, sandeepu, barsik}@us.ibm.com

***Abstract*—Modern storage systems are employing data deduplication with increasing frequency. Often the storage systems on which these techniques are deployed contain important data, and utilize fault-tolerant hardware and software to improve the reliability of the system and reduce data loss. We suggest that data deduplication introduces inter-file relationships that may have a negative impact on the fault tolerance of such systems by creating dependencies that can increase the severity of data loss events. We present a framework composed of data analysis methods and a model of data deduplication that is useful in studying the reliability impact of data deduplication. The framework is useful for determining a deduplication strategy that is estimated to satisfy a set of reliability constraints supplied by a user.**

I. Introduction

Data deduplication is increasingly being adopted to reduce the data footprint of backup and archival storage, and more recently has become available for near-line primary storage controllers. Scale-out file systems are increasingly diminishing the silos between primary and archival storage by applying deduplication to unified petabyte-scale data repositories spanning heterogeneous storage hardware. Cloud providers are also actively evaluating deduplication for their heterogeneous commodity storage infrastructures and ever-changing customer workloads.

While the cost of data deduplication in terms of time spent on deduplicating and reconstructing data is reasonably well understood [1], [2], its impact on data reliability is not, especially in large-scale storage systems with heterogeneous hardware. Since traditional deduplication keeps only a single instance of redundant data, it magnifies the negative impact of data loss. Chunk-based deduplication [3], [4] divides a file into multiple chunks, meaning the loss of one chunk will create many lost chunks in the storage system. Delta encoding [5], [6], [7], [8] deduplicates at the file level, storing the differences among files, and creating the potential for losing multiple files when ever a file is lost.

Administrators and system architects have found understanding the data reliability of their system under deduplication to be important but extremely difficult [9]. Deduplication itself poses two potential reliability problems. First as illustrated in Figure 1a, the fact that chunks in several files depend on a single instance means that loss of a instance causes secondary losses across the data store. Inversely, as shown in Figure 1b, files with multiple references to different chunks may be dependent on the reliability of multiple storage devices. If any one of the devices upon which it depends fails, the file itself is lost. This additional dependence may be counter-balanced, however, depending on the degree of additional storage efficiency. By decreasing the number of disks in the system with deduplication, we decrease the number of expected disk faults as well. Understanding these complex relationships requires understanding the nature of deduplication itself, as well as the complex interactions created by the underlying storage system.

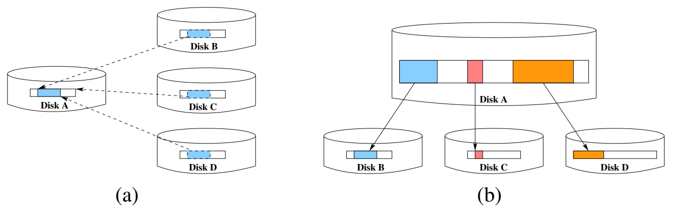


Fig. 1: Deduplication example showing the dependence of multiple references to the same chunk in multiple files to a single stored reference and in a file to multiple disks in the data store.

The existing literature relies on heuristics to address the issue of reliability in deduplication systems; the key recommendation is to keep multiple copies of a data chunk instead of storing only a single instance. The creators of Deep Store [4] proposed to determine the level of redundancy for a chunk based on a user-assigned value of importance. It has been suggested by D. Bhagwat et al. [3] that the number of replicas for a chunk should be proportional to its reference count, i.e., the number of files sharing the chunk. A gap exists in the current literature on the topic of quantifying the data reliability of a deduplication system or providing a means to estimate whether a set of reliability requirements can be met in a deduplication system.

A quantitative modeling of reliability in a deduplication system is nontrivial, even without taking into account the petabyte scale of storage systems. First, there are different types of faults in a storage system, including whole disk failures [10], latent sector errors (LSEs) [11], [12], and undetected disk errors [13], [14], [15]. To consider all these faults together, it is necessary to have an understanding of how these faults manifest, and have a representative model that takes into account dependencies and correlations with other, similar, faults as well as the interactions of independent faults in the hardware environment. Second, these faults can propagate due to the sharing of data chunks or chaining of files in a deduplication system. In order to correctly understand the impacts of these faults and their consequences on the reliability of our storage system, we need to accurately model both the storage system faults and faults due to data deduplication. We call storage system faults and faults due to deduplication *primary* and *secondary faults* respectively, and discuss them in more detail in Section IV. Third, it is important to note that many of the faults we wish to consider are rare compared to other events in the system, such as disk scrubbing, disk rebuilds, and I/O. Calculating the impact from rare events in a system can be computationally expensive, motivating us to find efficient ways of measuring their effect on the reliability metrics of interest.

The complexity of this problem arises from two different causes. The first is the state-space explosion problem which can make numerical solution difficult. As our model grows increasingly complex the state space grows rapidly. The simplified deduplication system studied in [16] quickly grows to unmanageable size, having 10^{23} states with only 10 storage subsystems, and 10^{222} states with 100 storage subsystems, exceeding the capabilities of numerical solvers. A second issue comes from the stiffness that results from rare events. For numerical solutions stiffness introduces numerical instability, making solution impractical. When simulating stiffness increases the number of events we must process, causing a resulting increase in simulation complexity. These factors, and a desire to precisely understand the complex relationships present a need to use more sophisticated methods of analysis to fully understand the implications of deduplication.

A. Our Contributions

In this paper, we utilize a discrete event simulation approach to quantitatively analyze the reliability of a modeled deduplication system with heterogeneous data on heterogeneous storage hardware, in the presence of primary faults and their secondary effects due to deduplication. The analysis is based on three key dimensions that our model takes into account:

- The fault tolerance characteristics of the underlying storage hardware.
- The statistical deduplication characteristics (e.g, reference count distribution) of a set of deduplicated data.
- The number of replicas of a given data chunk that are stored.

To validate our modeling approach, we studied data from an enterprise backup storage system containing 7 terabytes of deduplicated data comprising over 2.7 million unique file names and over 193 million references to 2.87 million unique deduplicated data chunks. We analyzed the statistical properties of this real data, including the deduplication relationship implied by references from each file to deduplicated chunks. To a user, different data sets usually have different importance and different reliability constraints. Treating all files the same way is not the right strategy. Therefore, we break our analysis out into twelve separate categories that are based on the file types and applications, characterizing each category separately.

We derived a model of the data that has the same statistical characteristics as the original data set, and evaluated the reliability impact of data deduplication on a variety of different storage hardware with different reliability characteristics, different data categories with different deduplication characteristics, and different numbers of replicas for deduplicated chunks.

The key contributions of this paper are:

- **Design and implementation of a modeling framework to evaluate the reliability of a deduplication system with varying hardware reliability characteristics, varying deduplication characteristics, and a varying number of copies of data chunks.** We built a complex reliability model based on the studied enterprise system and on models of faults in storage systems described in the literature. We evaluated the data reliability of our system under an ecosystem of faults composed of whole disk failures [10], latent sector errors (LSEs) [11], [12], and undetected disk errors [10], [13], [14], [15]. We explicitly modeled these faults as well as the faults resulting from data deduplication to measure the impact of all faults on the rate of both file loss and deduplicated chunk loss. Due to the rarity of the faults we propose to model (compared to other processes such as I/O and disk scrubbing), we utilized a simulation algorithm that exploits near-independent relationships [17] in the model to make solution of our model for fault tolerance metrics more efficient.
- **Analysis of the effect of multi-copy deduplication.** Our framework can also be used to determine the number of copies of various data chunks or files needed to meet a specific reliability requirement while minimizing storage utilization. For our system, we utilized category information along with reference counts to determine the importance of a deduplicated chunk. For our system we believe that this is preferable to methods based only on the reference count of the deduplicated chunk, as in [4], [3].

Our model is currently based on offline storage-side deduplication using variable chunk hashing. We believe it can be easily extended to support other deduplication algorithms, including fixed-size chunk hashing, whole file hashing, and delta encoding, as well as other deduplication architectures, including online storage-side, online storage-side, and online client-side. This extension would involve computing new empirical distributions for the underlying deduplication system, with few, if any, changes to our model.

B. Related Work

While other studies have approached the question of reliability in data deduplication, they tend to assess impact through an assumption that the number of files referencing a deduplicated chunk is directly proportional to the importance of a chunk [4], [3]. While this may be an appropriate assumption when studying data whose types are largely homogeneous [4], [3], we believe this provides a limited picture of how deduplication affects fault tolerance on real systems storing a large amount of heterogeneous data. Moreover, these studies [4], [3] do not provide a way to quantify the data reliability of a deduplication system.

Our study differs in that it quantitatively analyzes the reliability of a deduplication system with heterogeneous data and heterogeneous storage hardware. We use our methodology to provide insight to help meet design goals for reliability while maintaining an improved storage efficiency over a non-deduplicated system. Our quantitative analysis is performed utilizing discrete event simulation and by exploiting identified near-independent relationships in the underlying model to more efficiently solve a complex model in the presence of rare events as described in [16].

II. Overview of Our Reliability Analysis Framework

The modeling framework is composed of two main components: a deduplicated file system model and a hardware system reliability model. These models are solved using a discrete event simulator described previously [16]. Given the data to be stored in a deduplicated storage system and the deduplication algorithm used on the system, including parameters such as chunk size and similarity measures, we built a model of deduplication in our storage system that represents the resulting deduplication process. Data in our system are categorized into different classes based on either known application file extensions or user-specified criteria. Our deduplication model summarizes the relationships implied by deduplication for each of these classes. Section III describes the process by which we characterize data, along with its application on a real data set.

The reliability model of our hardware makes use of system level configuration information to build an on-line mathematical representation of our hardware environment. Specifically, parameters that have more impact on reliability are considered, such as the type of disks used in our system (nearline or enterprise), RAID type (1, 5, and 6) or erasure codes, the size of a stripe on our arrays of disks, and the number and configuration of disks in a reliability group. We model three types of storage-level faults explicitly, including whole disk failure, latent sector errors, and undetected disk errors. Secondary faults due to deduplication are deduced via our model of the deduplicated file system. An in-depth discussion of these topics is provided in Section IV.

Given the model of deduplication, the hardware reliability model, and the parameters of the target system (such as the replication factor and data distribution), our discrete event simulator provides per-category estimates of expected reliability for our hardware, deduplicated file system, and parameter sets. Section V summarizes the results of one such estimation on several hardware systems using a real-world data set and various deduplication parameters.

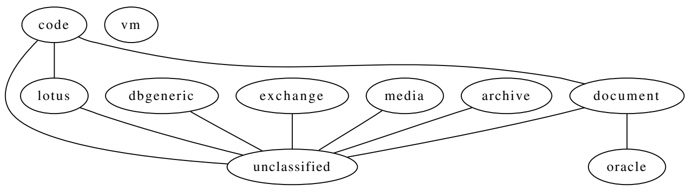


Fig. 2: Summary of categories that contain files that share deduplicated chunks with other categories.

III. Deduplicated File System Model

In order to evaluate the effects of data deduplication on fault tolerance in a real system, we examined the deduplicated data stored in an enterprise backup/archive storage system that utilizes variable-chunk hashing [18], [19]. Our example is a client/server system that provides backup and archive solutions for a multi-vendor computer environment that can consist of file servers, workstations, application servers, and other similar systems. Server-side data deduplication is used for space management in this system.

We present a general model of the relationships implied by data deduplication, and their consequences for fault tolerance, based on our analysis of the real system. We also present refinements to the model necessary to model variable-chunk hashing, as used by the system represented in our data. In order to demonstrate the flexibility of our model of deduplication, we will also show how to adapt it for delta encoding.

A. Data Analysis

The data stored in the system consist of backups from about 150 workstations (the most common operating system family for workstations in the backup group was Windows, though MacOS and Linux were also represented), as well as backups of several IBM DB2, Oracle, and Microsoft SQL database servers and several mail servers, including IBM Lotus Domino and Microsoft Exchange. The data-set has approximately 7TB of deduplicated data. Without deduplication, the system would require over 16TB to store all the data. The deduplicated data have a total of 193,205,876 separate references to 2,870,681 unique data chunks.

In order to better understand the deduplication relationships implied by our data, we placed all files on the system into eleven categories based on their file extensions. A total of 2,735,894 unique file names were processed, featuring 55,688 unique file extensions. Of these file extensions, only 14,910 appeared more than once, and only 1,520 appeared five times or more. We identified four major categories and eleven subcategories based on file extensions.

- *Databases*: We specified four categories for files associated with database applications **db2**, **Oracle**, **SQL** and **DBGeneric**. We use DBGeneric for those files we know

	Unique Chunks	References per Chunk	
		90th Quantile	Maximum
Archive	50,240	24	174,720
Code	895,615	2	105,404
Document	574,222	2	16,128
Exchange	9,288	4	42,442
Lotus	9,790	14	60,216
Media	148,887	4	3,384
MSSQL	16,089	32	280,044
Oracle	30,460	4	21,476
db2	30,810	6	5,194
DBGeneric	20,456	6	77,120
VM	9,328	2	308,934
Unclassified	1,075,851	8	251,542

TABLE I: Summary of the data obtained from analysis of deduplicated chunks.

to be used by a database, but for which the specific database is unknown.

- *Mail*: We identified two categories of files associated with mail applications: **Lotus Domino** and **Exchange**.
- *User Data*: We specified four categories for user data files: **Archives**, **Documents**, **Media** and **Code**.
- *VM*: We grouped all virtual machine application data into a single category, **VM**.

We call our twelfth category **Unclassified** and use it to hold system files we assume to be re-creatable from installation media or other sources that make recovery of the data possible, files whose extensions do not match expected for our previous eleven categories, and those files with no file extensions.

We do not suggest that these categories are the best or only ways to partition a deduplicated file system. In fact, we assert that the proper way to partition a file system into categories is context-sensitive and user-specific, based on legal and contractual obligations as well as administrator goals. Categories should reflect groups of important files or applications. To understand the relationships that these categories of files shared through deduplication, we constructed a graph with a set of nodes N_I with an element for every deduplicated chunk in our deduplicated system, and a second set N_C with a node for each category of file defined. When we encountered a reference in the data from a category to some deduplicated chunk, we added the edge connecting the nodes, allowing duplicate edges. The weight of an edge is equal to the number of duplicate edges and defines the number of references to a given deduplicated chunk.

Using this graph, we identified 351 chunks with references from exactly two categories, and two with references from exactly three categories. The remaining 2,870,328 chunks had references from only one category. Figure 2 shows the paths between nodes in N_C that pass through exactly one node in N_I and no nodes in N_C. It seems likely that the frequent connections between the unclassified node and other nodes in N_C, represented in Figure 2, are indicative of a failure to properly classify files by their extensions, or files with misleading extensions. In such cases, it seems safest to treat unclassified files that reference chunks that share an edge with another category C_k as if they are from category C_k. For those nodes shared between two categories C_i and C_j, where neither is the unclassified category, we consider the node a legitimate cross category deduplication. For our analysis we treat a deduplicated chunk as categorized with the highest level of importance of any referring file or category.

The distribution of references to chunks varied based on which categories were connected in our graph. Table I summarizes some of this information by showing the total number of unique deduplicated chunks with at least one reference to each of the categories, the 90th quantile for references per chunk for those chunks with at least one reference to a given category, and the maximum number of references per chunk for those chunks with at least one reference to a given category.

B. Using Category Information to Define Importance

Determining the importance of a file on a deduplicated storage system is a difficult proposition, and is likely to be dependent on the needs and legal requirements of the organization operating the storage system. While one could take the approach of keeping additional copies of all files that have a certain reference count, we suggest that this is a poor measure of importance, for two primary reasons. First, it assumes that chunks with few references are less important. While it is true that the loss of a chunk with fewer references will cause fewer secondary failures due to deduplication, deduplicated chunks fail at the same rate regardless of their reference count, and those chunks with fewer references may be referenced by critical data whose reliability is just as important as that of files that share a chunk with many other files. Second, using reference count as a measure of importance can result in a loss of storage efficiency to increase the reliability of files that are either easily re-creatable system files, or files unimportant to the organization.

C. Model of Deduplicated File System

We view deduplication on this file system as a dependence relationship and construct a graph, whose nodes represent files and deduplicated chunks in our file system, to model this dependence relationship. Each deduplicated chunk in our file system is represented by a node $n_i \in N_I$. Files themselves are represented by the set $N_F = \{N_{F,C_1}, N_{F,C_2}, N_{F,C_3}, \ldots, N_{F,C_{12}}\}$, where each subset N_{F,C_k} contains a node n_{j,C_k} for each file f_j that is a member of category C_k. Deduplication relationships are again represented by the set of edges E such that if a chunk n_i is referenced by a file f_j in category C_k, an edge $n_{j,C_k}n_i \in E$.

We suggest using the data summarized in Table I as an empirical estimate of the probability density function (pdf) for a random variable representing the number of references for a chunk in the given category c. Using this pdf, $f_c(x)$, we define an inverse distribution function (idf) $F_c^* : (0,1) \rightarrow \mathcal{X}$, defined for all $u \in (0,1)$ as follows:

$$F_c(x) = Pr(X \leq x) = \sum_{t \leq x} f_c(t) \quad (1)$$

$$F_c^*(u) = \min_x \{x : u < F_c(x)\} \quad (2)$$

Using $F_c^*(u)$ and a uniform random variate $\mathcal{U}$, we can generate realizations of the random variable described by

$f_c(x)$, allowing us to use our observations summarized in Table I to synthetically create a deduplication system with the same statistical properties as our example system. The number of edges connecting to any node $n_i \in N_I$ is defined by first determining the primary category of files that refer to the chunk, and by using Equation 2 to generate a realization of the random variable described by $f_{C_k}(x)$.

While our study concerns only data deduplication that uses variable-chunk hashing, it is a simple matter to adapt this model for delta encoding or whole file hashing. In those cases, we simply remove the set N_I and define edges between elements of N_F directly. Then the edges in the E must be represented as directed edges of the form $\overrightarrow{n_{f_a} n_{f_b}}$, indicating that the file n_{f_a} depends on n_{f_b}. Directed edges are not required for our variable-chunk hashing representation, as it is implied that the relationship is a dependence of nodes in N_F on nodes in N_I.

IV. Hardware Reliability Models

Traditional disk failure encompasses those faults accounted for by the manufacturer when calculating Mean Time To Failure (MTTF) [10]. We assume that these traditional disk failures are detected by the disk immediately when they occur, and require no special detection mechanisms. Traditional disk failures are assumed to be non-transient and unrepairable without drive replacement.

Another kind of primary fault that we model are latent sector errors (LSEs). LSEs differ from traditional disk failure in that they cannot be detected until the corresponding disk sectors are accessed. LSEs can be either transient or permanent [11]. An LSE is typically understood to be a condition in which a given disk sector cannot be read or written, or when there is an uncorrectable ECC error. It is important to note that even in the case of a transient LSE, previous study of LSEs has indicated that data stored in the sector are irrevocably lost, even when the sector can later be read or written to properly [11]. Latent sector errors have been found to be correlated in both space and time as described by Schroeder, Damouras and Gill in [12], who demonstrate these correlated LSEs as bursts best characterized by a Pareto distribution. These bursts of correlated LSEs represent the fact that disks experiencing a single LSE are likely to suffer from a series of LSEs in the near future, and those LSEs are likely to be correlated spatially on the disk with previous errors. In our system model, we consider LSEs to be correctable either when the disk is subsequently rebuilt due to a traditional disk failure, or upon performance of a scrub of the appropriate disk.

Our third and final type of primary fault is that of undetected disk errors. UDEs represent silent data corruption on the disk, which is undetectable by normal means [20], [13], [14]. UDEs are drawn from two distinct classes: *undetected read errors* (UREs) and *undetected write errors* (UWEs).

UREs manifest as transient errors, and are unlikely to affect system state after their occurrence. They represent cases in which incorrect sectors are read and returned to the user instead of the proper data, or those cases where the read head passes too high over the platter to read the correct data. UWEs are persistent errors that are only detectable during a read operation subsequent to the faulty write.

UWEs can be further subdivided into three types: *dropped writes*, *near-off-track writes*, and *far-off-track writes*. In the case of dropped writes and near off-track writes, only the data in the target sectors are corrupted. Far-off-track writes corrupt local data as well as data on other parts of the disk. UREs can be similarly subdivided into near-off-track reads and far-off-track reads. Off-track writes (both near and far) occur when the write head is not properly aligned with the track. In the case of a near-off-track write, the data is written in the gap between tracks, adjacent to the intended track. On a read operation, the head may align itself to either the target track or the off-track, potentially producing stale data. Far-off-track writes occur when data are written even further off-track, such that they corrupt data in another track entirely. Subsequent reads to the track that was mistakenly written to will produce corrupt data, while reads to the track that should have been written to will produce stale data [14]. We consider UDEs to be correctable when the disk is rebuilt because of a traditional disk failure, upon performance of a scrub of the appropriate disk, or when the error is overwritten before being read, although this type of mitigation produces parity pollution [14].

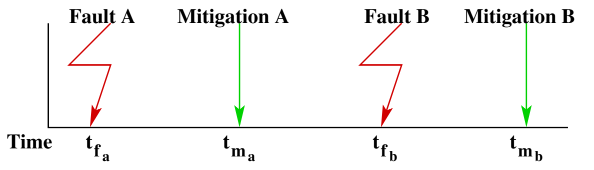
A. Disk Model

In order to understand the effect of faults in an example system, we utilize a formal model of disks in our underlying storage system. Each disk in our system is represented as a 5-tuple, $D_i = \{\varXi, F_\ell, F_\upsilon, F_\phi, f\}$. The elements of D_i define the current fault state of the disk D_i. The variable $\varXi$ defines the set of all contiguous natural numbers between $[0, n_{D_i} - 1]$, where n_{D_i} is the number of sectors on the disk. All faults on a given disk are defined over this set, using it to represent the physical subdivisions of the disk. Faults on the disk are given by the sets $F_\ell, F_\upsilon, F_\phi$ and the scalar f. We utilize $F_\ell = \{\ell_0, \ell_1, \ldots\}$ to represent the set of all latent sector errors (LSEs) currently affecting the drive, $F_\upsilon = \{\upsilon_0, \upsilon_1, \ldots\}$ to represent the set of all UWEs currently affecting the drive, $F_\phi = \{\phi_0, \phi_1, \ldots\}$ to represent the parity strips on the disk that have been polluted due to a parity pollution event, and the scalar $f = \{0, 1\}$ to indicate whether the disk has suffered an entire disk failure. Members of the sets $F_\ell, F_\upsilon, F_\phi$ are defined over $\varXi$ to represent the portions of the disk that have suffered LSEs, UDEs, or polluted parity.

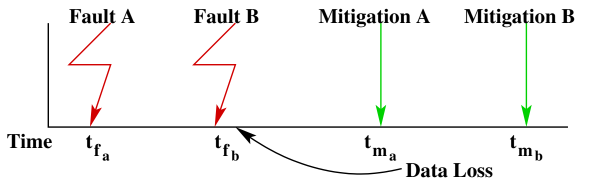
Disks are gathered in the model into sets of m disks, $G_i = \{D_j, D_{j+1}, \ldots, D_{j+m}\}$, representing RAID groups. In the case of RAID 5 groupings with three data disks and one parity, each group G_i in the system would contain four disks.

B. Fault Interactions and Data Loss

It is important to note that the occurrence of a fault within our system does not guarantee that data loss has occurred. In many cases, the underlying storage system will utilize some form of fault tolerance, such as RAID. For that reason it is



(a) In the case when subsequent fault events arrive to the system after a mitigation event for all previous faults has been processed, there is no potential interaction.



(b) Subsequent faults that arrive to the system before mitigation events for previous faults have the potential to result in data loss.

Fig. 3: Example fault interactions.

important to separate the modeling of faults and errors in our model. For the purposes of our model we consider faults to include traditional disk failure, LSEs and UDEs, and errors to include data loss which cannot be recovered, and serving corrupted data silently to the user. In general, for a fault to manifest as a data loss error, we must experience a series of faults within a single RAID unit. How these faults manifest as errors depends on the ordering of faults and repair actions in a time line of system events, as shown in Figure 3. In the case of RAID 5, a single failure can usually be tolerated before a data loss event occurs. For RAID 6, two failures can be tolerated before data loss. UDEs cause a different kind of error, which is largely orthogonal to RAID, by silently corrupting data which can then be served to the user.

In order to determine if a combination of primary faults has led to data loss and potentially a number of secondary faults, we examine the timing of events in a manner similar to the window of vulnerability method described by [21]. Given a storage system that can tolerate n faults before data loss occurs, we will see faults manifest as data loss only when the joint effect of n faults occurs on overlapping portions of disks in the same reliability group before mitigation. To evaluate that, we utilize the representations of the faults on the disk as defined in Section IV-A.

Faults in the form of traditional disk failures can result in data loss if their arrival times t_{f1}, t_{f2} are such that for the time at which the initial fault is mitigated $(m_{f1} > t_{f2}) \wedge (m_{f2} > t_{f2})$. In such a case, the entire drive is lost to the failure.

Traditional disk failures can also result in data loss when combined with a subsequent LSE on a read operation. Again, given arrival times of the failure events t_{f1}, t_{f2} and a mitigation time for the first fault m_{f1} $(m_{f1} > t_{f2}) \wedge (m_{f2} > t_{f2})$, an LSE on another disk in the RAID group that corrupts data on the disk before mitigation will result in the rebuilding of an unrecoverable sector on the disk.

UDEs form a special case of fault. While they can be detected by a scrub operation, repair is not possible. Scrubbing a disk tells us that an error is present in the stripe, but not where the error is. An error in the parity or on any of the data drives creates an identical situation for the scrub.

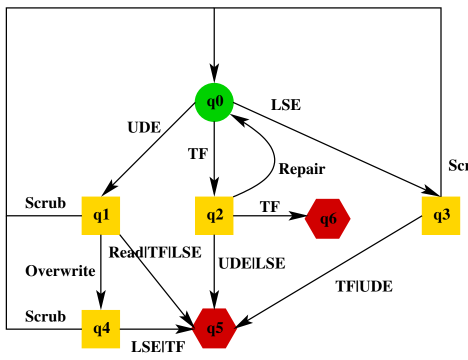


Fig. 4: DFA representing the combination of faults which lead to data loss on a stripe from UDEs, LSEs, and traditional failures under RAID1, or RAID5.

In order to characterize the interactions of faults in our model, we maintain a state-based model of portions of the physical disk, as represented by $\Xi \in D_i$ from Section IV-A. Given a set of disks that are grouped into an interdependent array (such as the set of disks in a RAID5 configuration, or a pair of disks that are mirrored), each stripe in the array maintains its state using a state machine appropriate to the number of tolerated faults the configuration can sustain without data loss, such as shown in the example in Figure 4.

Each stripe is represented by a tuple $(Q, \Sigma, \delta, q_0, F)$. The set of states Q can be partitioned into three different subsets; $Q_{good} = \{q_0\}$, the fault-free non-degraded state and start state; $Q_{degraded} = \{q_1, q_2, q_3, q_4\}$, states in which the stripe has suffered a fault but no data loss; and $Q_{fail} = F = \{q_5, q_6\}$, which represent the states that indicate that data have been lost. When the simulator processes an event for a given stripe, it forwards information on the processed event to the state machine in the form of the DFAs input alphabet, $\Sigma = \{TF, LSE, UDE, Write, Read, Scrub, Repair\}$. Each of those symbols represents a fault, a mitigation, or an action that causes a UDE to serve corrupt data undetectably. The DFA transitions on these symbols based on the transition relation defined by $\delta : Q \times \Sigma \rightarrow Q$.

The DFAs maintained by stripes within our modeled system are generated automatically using knowledge of potential fault interactions and parameters that define the size of the disk array s_{array} and the number of disk faults $n_{\text{tolerated faults}}$ that the array can tolerate without data loss as defined by the array's RAID level [22].

The set of all DFAs, combined with our model of deduplication, the system clock, and models of fault correlation for "active" LSE bursts, comprises the state of our model, which we then proceed to solve via discrete-event simulation [23], [24], [25]. Events are generated in a state dependent manner for all potential fault and mitigation actions in the system.

After each event is processed, fault metrics are then calculated through checking of the state of each stripe. For computational efficiency, these states are stored in a sparse array, where the default state of each DFA is assumed to be

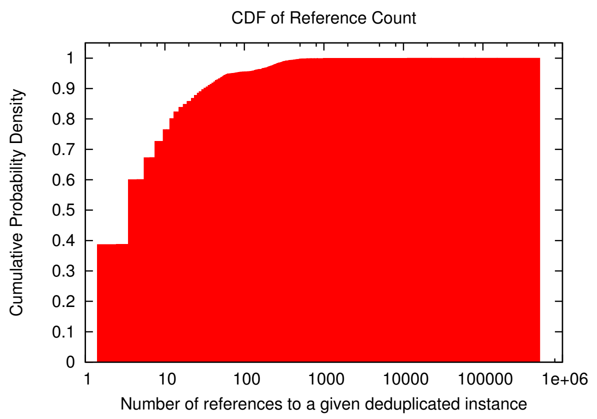


Fig. 5: Cumulative probability density function of the number of references to each deduplicated instance for the SQL category.

q_0. Those DFAs found to be in an accepting state are faulty, with state q_5 indicating that one or more stripes have failed, and q_6 indicating that entire drives have been lost. A list of all damaged stripes is calculated. Any files on those stripes are categorized and reported lost. Any deduplicated chunks n_i on those stripes are categorized and reported as lost, and secondary faults are generated, reporting lost all files share references to the lost file.

C. *Deduplication Model*

When modeling the impact of errors in our system we utilize a model of deduplication based on the empirical data and analysis from Section III. If a primary fault causes a deduplicated chunk $n_i \in I$ to be lost, $\forall n_j \in N_F$ such that $\exists n_i n_j \in E$ the files represented by nodes meeting the criteria for n_j suffer a secondary fault due to the initial failure. These secondary faults would not have occurred, were it not for the deduplication strategy used by the storage system. In the case of whole file hashing, delta encoding, or other deduplication methods that allow for a chain of references, the loss of a file $n_i \in N_F$ implies not only the loss of all $n_j \in N_F$ such that $\exists \overrightarrow{n_j n_i} \in E$, but also all $n_{j+1} \in N_F$ such that $\exists \overrightarrow{n_{j+1} n_j} \in E$ recursively for files for which a path exists to the original lost file n_i. In the case of the loss of a disk segment containing a file, no secondary faults are triggered.

V. Discrete Event Simulation Results

In order the understand the impact of data deduplication on fault tolerance we simulated systems with data sets of 7TB (based on the system described in Section III) and 1PB before deduplication. Both systems are assumed to have a deduplication ratio of 0.5. We modeled the systems with reliability provided by various RAID and erasure codes, including RAID1 (mirroring), RAID5 in $7+p$ and $8+p$ configurations, RAID6 in an $8+2p$ configuration, and erasure codes in an $8+3p$ configuration. For each system we calculated two reliability measures: the rate at which all copies of a deduplicated chunk were lost, and the rate at which undetected corrupt data was served to applications.

We make the assumption that our modeled system features a workload of 100 io/s for each data disk in the system with reads making up 95% of the workload. Disks themselves are assumed to be 750GB with 128k strips, with read and write requests simulated only for those portions of a disk containing data. We assume data is distributed uniformly across all disks, and that reads and writes are likewise uniformly distributed. We derive the rate of traditional disk failures from [10], and latent sector errors from [12] using the parameters given for system $A-1$ in the paper. We derive rates for UWEs are the same as described in [15] for enterprise drives.

Reliability is effected by deduplication in two ways: the *incidence* and *impact* of faults. We found in our simulations, that the incidence of faults is reduced by a factor equal to the deduplication ratio, due to the reduction in the number of disks required to store the same data set. The impact, however, of each fault was increased for all categories of data by a factor larger than the reduction provided by deduplication, resulting in a net decrease in fault tolerance for all categories of files. The impact of a fault during simulation was calculated using empirical cumulative probability distributions calculated from the data set. First the file or files suffering an error were assigned a category randomly, next based on the category, we randomly determined whether the segment of the file or files lost contained a deduplicated chunk based again on empirical distributions from our data. Finally, if the file lost was a deduplicated chunk we generated a random number of references to the chunk, and assigned them randomly generated locations in the storage system so that in the case of errors spanning multiple files, a file and adjacent reference were not double counted when both were lost to the initial error. An example CDF for the number of references to a deduplicated for files in the SQL category is shown in Figure 5.

The rate of permanent data loss due to unrecoverable faults is shown in Tables II and III for RAID1 and RAID5. The rate of loss increases for all categories when only a single instance is kept per deduplicated chunk. For configurations with higher fault tolerance (RAID6 and $8+3p$ erasure codes), we saw no significant decrease in fault tolerance during the expected lifespan of a typical storage system. Any increase in the impact of data deduplication on the unrecoverable loss of system data is masked by the low incidence of unrecoverable data loss during the expected system lifespan in the systems we studied.

The dramatic improvements witnessed when two copies are kept of each deduplicated chunk are to do to the circumstances which must occur in order to permanently lose the data stored in the chunk. In addition to the requisite correlated faults shown in Figure 3, the same situation must occur on the independent storage unit which holds the other copy of the instance before the first storage unit is restored. When the correlated faults involve a LSE, the situation becomes even more unlikely, requiring the other copy of the deduplicated chunk to not only reside on the same disk as the error, but the same stripe.

	7 TB			
	RAID1		RAID5	
	1 copy	2 copy	1 copy	2 copy
Archive	1.2e+02 ± 6.2e+01	8.6e-10 ± 2.2e-10	1.7e+03 ± 7.1e+02	1.8e-07 ± 2.9e-0
Code	2.9e+03 ± 6.4e+02	2.8e-08 ± 1.3e-09	6.8e+04 ± 3.0e+04	1.5e-05 ± 2.9e-06
db2	3.3e+03 ± 1.6e+03	5.4e-08 ± 1.3e-08	4.1e+04 ± 2.2e+04	8.6e-06 ± 2.5e-06
DBGeneric	6.5e+01 ± 8.5e+01	1.3e-09 ± 2.3e-09	1.9e+02 ± 1.3e+02	1.2e-08 ± 5.5e-09
Document	6.6e+01 ± 1.1e+02	1.3e-09 ± 3.3e-09	2.2e+02 ± 1.1e+02	1.4e-08 ± 3.5e-09
Exchange	4.0e+02 ± 1.5e+02	3.2e-09 ± 4.1e-10	5.6e+03 ± 1.3e+03	6.1e-07 ± 3.3e-08
Lotus	2.4e+01 ± 2.1e+01	1.1e-10 ± 7.6e-11	4.0e+03 ± 5.2e+03	2.8e-06 ± 5.0e-06
Media	1.3e+03 ± 2.6e+03	1.7e-07 ± 6.4e-07	1.3e+03 ± 8.7e+02	1.7e-07 ± 7.2e-08
Oracle	7.2e+01 ± 4.1e+01	7.4e-10 ± 2.4e-10	4.8e+02 ± 2.8e+02	3.3e-08 ± 1.1e-08
SQL	7.2e+01 ± 4.1e+01	4.9e-10 ± 1.6e-10	1.2e+03 ± 6.8e+02	1.3e-07 ± 4.4e-08
Unclassified	4.9e+03 ± 1.5e+03	6.5e-08 ± 6.3e-09	1.6e+05 ± 1.8e+05	7.0e-05 ± 8.4e-05
VM	2.6e+01 ± 2.7e+01	2.1e-10 ± 2.3e-10	2.7e+02 ± 1.1e+02	2.3e-08 ± 4.0e-09

TABLE II: Estimated rate of file loss per year, for the 7TB system using RAID1 and RAID5, and a single copy of each deduplicated chunk.

	1 PB			
	RAID1		RAID5	
	1 copy	2 copy	1 copy	2 copy
Archive	8.2e+03 ± 4.1e+03	5.7e-08 ± 1.5e-08	1.2e+05 ± 4.7e+04	1.2e-05 ± 1.9e-06
Code	2.0e+05 ± 4.3e+04	1.9e-06 ± 8.8e-08	4.6e+06 ± 2.0e+06	1.0e-03 ± 1.9e-04
db2	2.2e+05 ± 1.1e+05	3.6e-06 ± 8.9e-07	2.8e+06 ± 1.5e+06	5.7e-04 ± 1.7e-04
DBGeneric	4.3e+03 ± 5.7e+03	8.7e-08 ± 1.5e-07	1.3e+04 ± 8.9e+03	7.9e-07 ± 3.7e-07
Document	4.4e+03 ± 7.0e+03	8.5e-08 ± 2.2e-07	1.4e+04 ± 7.3e+03	9.1e-07 ± 2.3e-07
Exchange	2.7e+04 ± 9.7e+03	2.1e-07 ± 2.8e-08	3.7e+05 ± 8.7e+04	4.1e-05 ± 2.2e-06
Lotus	1.6e+03 ± 1.4e+03	7.0e-09 ± 5.1e-09	2.6e+05 ± 3.5e+05	1.9e-04 ± 3.3e-04
Media	8.8e+04 ± 1.7e+05	1.1e-05 ± 4.2e-05	8.9e+04 ± 5.8e+04	1.1e-05 ± 4.8e-06
Oracle	4.8e+03 ± 2.7e+03	4.9e-08 ± 1.6e-08	3.2e+04 ± 1.9e+04	2.2e-06 ± 7.5e-07
SQL	4.8e+03 ± 2.7e+03	3.3e-08 ± 1.1e-08	8.0e+04 ± 4.6e+04	8.9e-06 ± 2.9e-06
Unclassified	3.3e+05 ± 1.0e+05	4.4e-06 ± 4.2e-07	1.1e+07 ± 1.2e+07	4.7e-03 ± 5.6e-03
VM	1.7e+03 ± 1.8e+03	1.4e-08 ± 1.5e-08	1.8e+04 ± 7.6e+03	1.5e-06 ± 2.7e-07

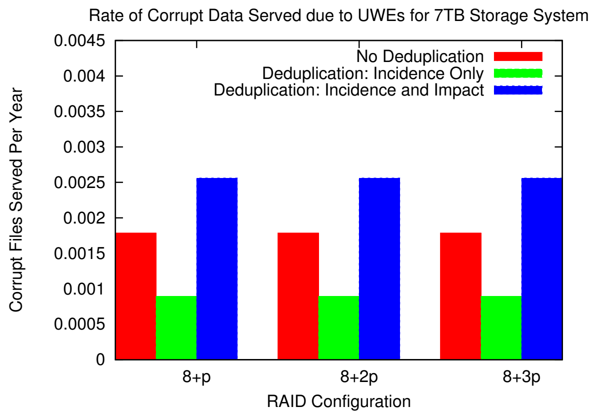
TABLE III: Estimated rate of file loss per year, for the 1PB system using RAID1 and RAID5, and a single copy of each deduplicated chunk.

For those systems which did suffer unrecoverable data loss, we kept a tally of the error scenarios leading to unrecoverable loss. All witnessed data loss events involved at least one disk failure. More than 50% also contained a LSE, while less than 2% contained a UWE. The high proportion of LSEs contributing to unrecoverable data loss stems from the temporal locality described by [12]. Unrecoverable data loss usually occurred during a campaign of LSEs coupled with a drive failure in the effected RAID unit (66.9% of the time), or an failure of two drives in a RAID unit before a rebuild could be accomplished (31.8% of the time).

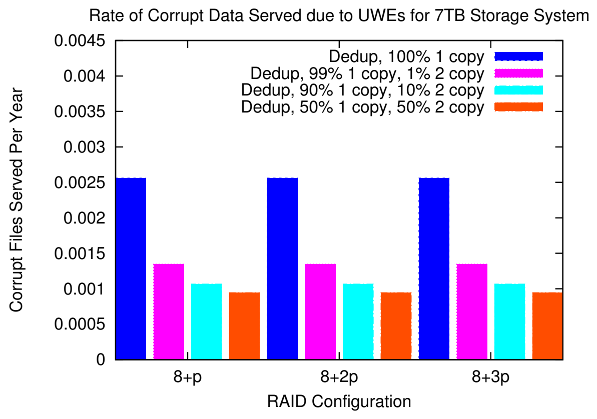
The decrease in fault tolerance due to data deduplication is easily offset by maintaining multiple copies of each deduplicated instance. Keeping as few as one additional copy results in a system more fault tolerant than the original, while still resulting in a mean increase in storage efficiency for all categories. It is important, however, to ensure additional copies are kept on separate RAID units to reduce the chance of correlated losses.

Multi-instance data deduplication maintains more than one copy for a distinct data chunk. It increases the resiliency of the system by orders of magnitude at the cost of increased space usage. The performance characteristics of such a system, i.e. write characteristics (data injection) and read characteristics (data reconstruction) are highly dependent on the architecture of the system, specifically the architecture of the meta-data manager. Unlike traditional single instance deduplication systems where hash maps or indices maintain data-signature to data-location mappings, in multi-instance deduplication systems, these bookkeeping data structures now have to accommodate more complex mappings. Further, managing (create/use/delete) these complex mappings adds overhead to both the CPU and IO.

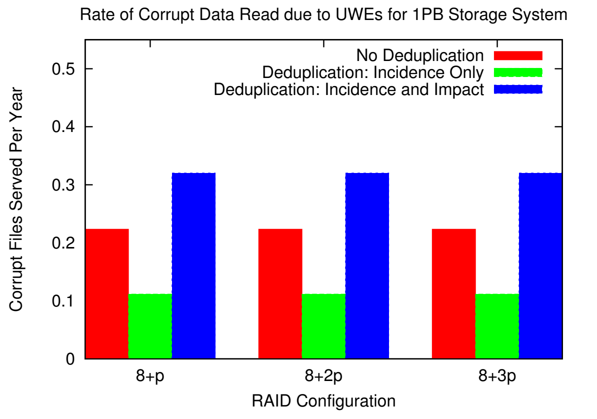
While UWEs did not significantly contribute to unrecoverable data loss, it is important to remember that they are fail silent, and largely orthogonal to RAID [14], [15]. When a UWE occurs on a system, it can cause corrupted data to be silently served when requested, before a disk scrub corrects the error. Figures 6a and 6d show the difference in the rate of corrupt data served for three different storage configurations for our 7TB and 1PB systems respectively for a sample data category. The first bar for each RAID configuration shows the rate of corrupt data being read for a non-deduplicated system. The second bar, to illustrate the different effects of incidence and impact, shows the different incidence only. The third bar shows the full effect of both incidence and impact. While the incidence of corrupt data is reduced, i.e. the smaller amount of data stored results in a lower number of corrupted files, the



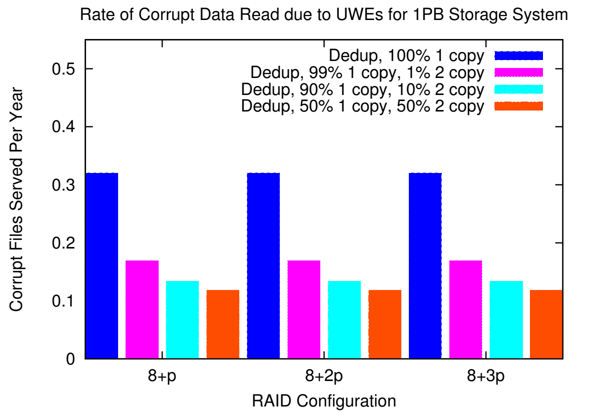
(a) Corrupted data served for systems with no deduplication, and with a single copy of each deduplicated chunk.



(b) Corrupted data served for systems with a single copy of each deduplicated chunk, and with two copies kept for the 1%, 10%, and 50% most referenced chunks.



(c) Corrupted data served for systems with no deduplication, and with a single copy of each deduplicated chunk.



(d) Corrupted data served for systems with a single copy of each deduplicated chunk, and with two copies kept for the 1%, 10%, and 50% most referenced chunks.

Fig. 6: Rate of undetected corrupted reads for the SQL category, for systems with data sets of size 7TB and 1PB.

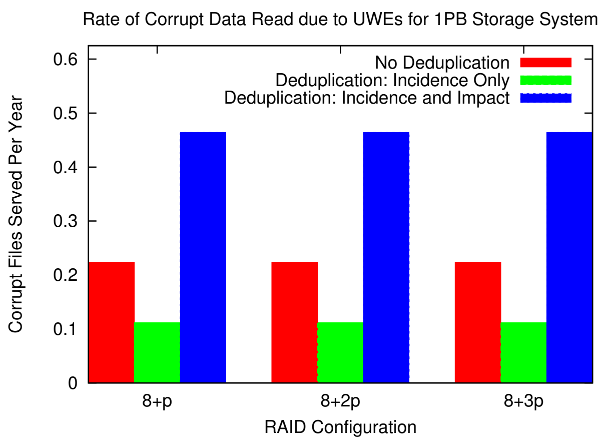
increased number of references results in a higher incidence of corrupted data being served. Not only are reads to the corrupted file effected, but any read to a referring file will result in silently serving corrupt data to the user, increasing the overall rate of corrupted reads to the system due to a UWE.

Again, we find a solution by keeping multiple copies of each deduplicated instance. Figures 6b and 6d compare the rate of corrupted data being read of deduplicated systems with a single copy of each instance, to systems which keep two copies for a fraction of all files in a category. The graph shows the results for keeping two copies for the 1%, 10% and 50% of files within a category containing the largest number of references. In the case of the SQL category, this results in large improvements for just the top 1% most referenced files. Figure 7 shows similar results for the VM category. While the general trends are the same, it is useful to note that the results of keeping additional copies is dependent on the category. Unlike the SQL category, increasing the portion of the category which maintains multiple copies from 1% to 10% and then again to 50% does not provide a significant reduction in the rate of corrupt data served, due to a small number of deduplicated instances accounting for a large number of the references within the category. This highlights the importance of a detailed analysis, using category information when making assumptions about the underlying deduplicated system.

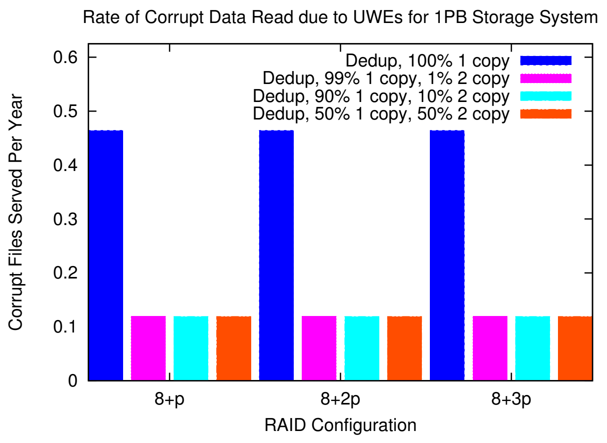
VI. Conclusions

Our evaluation of the effect of deduplication on our example system leads us to conclude that deduplication has a net negative impact on reliability, both due to its impact on unrecoverable data loss, and the impact of silent data corruptions, though the former is easily countered by using higher level RAID configurations. In both cases, system reliability can be increased by maintaining additional copies of deduplicated instances, and for the categories identified in our example system, typically by keeping multiple copies for a very small percentage of the deduplicated instances in a given category.

Our results emphasize the importance of detailed analysis of deduplicated systems to fully understand the impact of deduplication on fault tolerance. Even within our example system, individual categories features very different distributions,



(a) 1PB system, with only a single copy of each deduplicated instance.



(b) 1PB system, with multiple copies kept for some files.

Fig. 7: Rate of undetected corrupted reads for the VM category.

resulting in differing behaviors and trade-offs for multi-copy deduplication. Reliability returns decrease sharply for the VM category with increased proportions stored as multiple copies, due to the high portion with only a few references. For the VM category 90% had two or fewer references. Conversely, only 38% of deduplicated instances in the MSSQL category had two or fewer references.

While data deduplication helps to achieve goals of storage efficiency, its increasing prevalence raises legitimate reliability concerns. Given the increased regulatory pressure, and a desire to meet customer requirements for long-term data integrity, it is important to develop a further understanding of the reliability consequences of these methods.

For our example system, we show that reliability goals can still be met while maintaining some of the storage efficiency provided by deduplication by storing multiple copies of a portion of deduplicated instances. Our methodology could be applied to other systems to generate similar evaluations, and to evaluate configurations to meet design goals for both reliability and storage efficiency.

ACKNOWLEDGEMENT

This material is based upon work supported by the IBM PhD Fellowship program. The authors would like to thank Jenny Applequist for her editorial comments, as well as Kristin Rozier and Sankalp Singh for their helpful comments on earlier drafts of this paper.

REFERENCES

[1] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *USENIX FAST*, 2008, pp. 1–14.
[2] C. Ungureanu, B. Atkin, A. Aranya, S. Gokhale, S. Rago, G. Calkowski, C. Dubnicki, and A. Bohra, "HydraFS: A high-throughput file system for the HYDRAstor content-addressable storage system," in *FAST*, 2010, pp. 225–238.
[3] D. Bhagwat, K. Pollack, D. D. E. Long, T. Schwarz, E. L. Miller, and J.-F. Pris, "Providing high reliability in a minimum redundancy archival storage system," in *IEEE MASCOTS*, 2006, pp. 413–421.
[4] L. L. You, K. T. Pollack, and D. D. E. Long, "Deep store: An archival storage system architecture," in *ICDE*. IEEE, 2005, pp. 804–815.
[5] M. Ajtai, R. Burns, R. Fagin, D. D. E. Long, and L. Stockmeyer, "Compactly encoding unstructured inputs with differential compression," *JACM 49*, pp. 318–367, 2002.
[6] F. Douglis and A. Iyengar, "Application-specific delta-encoding via resemblence detection," in *USENIX ATEC*, 2003.
[7] L. You and C. Karamanolis, "Evaluation of efficient archival storage techniques," in *IEEE/NASA Goddard MSST*, 2004.
[8] J. MacDonald, "File system support for delta compression," Master's thesis, UC, Berkley, 2000.
[9] L. Freeman, "How safe is deduplication," NetApp, Tech. Rep., 2008. [Online]. Available: http://media.netapp.com/documents/tot0608.pdf
[10] B. Schroeder and G. A. Gibson, "Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?" in *FAST*, 2007, p. 1.
[11] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," *SIGMETRICS 35*, no. 1, pp. 289–300, 2007.
[12] B. Schroeder, S. Damouras, and P. Gill, "Understanding latent sector errors and how to protect against them," in *FAST*, 2010, pp. 71–84.
[13] A. Krioukov, L. N. Bairavasundaram, G. R. Goodson, K. Srinivasan, R. Thelen, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dussea, "Parity lost and parity regained," in *FAST*. USENIX, 2008, pp. 1–15.
[14] J. L. Hafner, V. Deenadhayalan, W. Belluomini, and K. Rao, "Undetected disk errors in RAID arrays," *IBM J Research and Development 52*, no. 4, pp. 413–425, 2008.
[15] E. Rozier, W. Belluomini, V. Deenadhayalan, J. Hafner, K. K. Rao, and P. Zhou, "Evaluating the impact of undetected disk errors in RAID systems," in *DSN*, 2009, pp. 83–92.
[16] E. W. D. Rozier and W. H. Sanders, "Dependency-based decomposition of systems involving rare events," Coordinated Science Laboratory, Univ. of Illinois, Tech. Rep. UILU-ENG-11-2203-CRHC-11-03, 2011.
[17] G. Ciardo and K. S. Trivedi, "A decomposition approach for stochastic reward net models," *Performance Eval. 18*, no. 1, pp. 37 – 59, 1993.
[18] M. O. Rabin, "Fingerprinting by random polynomials," Tech. Rep., 1981.
[19] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *CPM*. Springer, 2000, pp. 1–10.
[20] L. N. Bairavasundaram, G. R. Goodson, B. Schroeder, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dussea, "An analysis of data corruption in the storage stack," in *FAST*. USENIX, 2008, pp. 1–16.
[21] M. Baker, M. Shah, D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, T. Giuli, and P. Bungale, "A fresh look at the reliability of long-term digital storage," in *EuroSys, ACM SIGOPS*. ACM, 2006, pp. 221–234.
[22] D. A. Patterson, G. A. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," EECS Department, UC Berkeley, Tech. Rep. UCB/CSD-87-391, 1987. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/1987/5853.html
[23] L. M. Leemis and S. K. Park, *Discrete-Event Simulation: A First Course*. Prentice-Hall, 2005.
[24] W. D. Kelton, "Simulation analysis," in *WSC*. IEEE, 1983, pp. 159–168.
[25] J. Banks, J. Carson, B. L. Nelson, and D. Nicol, *Discrete-Event System Simulation*. Prentice-Hall, 2004.