THE UNIVERSITY OF CHICAGO

DATA SURVIVAL UNDER CATASTROPHIC FAILURES

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY

HUAN KE

CHICAGO, ILLINOIS
DECEMBER 2020

To my parents and my husband

Failures cannot be avoided, but data loss can.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Thanks to everyone who has been part of my PhD's journey. Primarily I would like to thank my advisor, Prof. Haryadi S. Gunawi, for his invaluable supervision and patience during my time at UChicago. Thanks for guiding me to explore various research topics, having good habits, and never ignoring the details. Without his persistent guidance and support, this dissertation would not have been possible. It is my great pleasure to start the PhD journey under his supervision.

Next, I would like to thank Prof. Hank Hoffmann for being on the committee with guidance and feedback. I am extremely fortunate to start my first class and finish my last defense with the great help from Hank. I also would like to thank Prof. Shan Lu who gave me guidance and feedback when I struggled with blockchain research. Thanks for teaching me the way to organize a study paper and guiding me to overcome the difficulties in research and life. Meanwhile I would like to thank the support and help from UCARE students Tanakorn Leesatapornwongsa, Riza Suminto, Cesar Stuardo, Mingzhe Hao, Huaicheng Li, Jeffrey Lukman, Meng Wang, Daniar Kurniawan, Michael Tong, and Shiqin Yan. Thanks for the academic support, cooperation, discussion, and great friendship.

One of the greatest pleasures of the PhD journey is to work with the Los Alamos National Laboratory (LANL) supervisor, Brad Settlemyer, who exposed the discipline of research to me with a lot of guidance and freedom. Thanks for teaching me the way to explore ideas and the skills to write papers. There are no words to express my gratitude for what he has done. Thanks for Los Alamos National Lab (LANL), where I grew to be professional with great help from David Bonnie, Nathan DeBardeleben, Michael Grosskopf, Elisabeth Moore, HB Chen, Jason Lee, and Naga. In particular, I would like to thank Brian Atkinson for teaching me ZFS, Lei Cao for running experiments, Terry Grov for guiding me doing presentation, and both Dominic Manno and David Bonnie for research discussions and paper editing. Most importantly, I would like to thank John Bent, who changed my life with supervision, encouragement, and trust in my research.

Outside research, there are always a few people who make my PhD life more memorable. I

would like to thank Shu Wang, Junwen Yang, Alex Enze Liu, and Wei Yuan who witnessed my ups and downs, always supporting and comforting me. I also would like to thank Camille Salerno for helping me out of sedentary lifestyle and taking me to restaurants, grocery stores, shopping malls, and arboretums. Thanks for exposing me to a healthier life with more vegetables, fruits, and exercises. Thanks for appearing in my life and bringing me more happiness than expected.

Back to the start, I would like to thank the support of my family. Thanks for always being there and providing me the freedom and chance to pursue dreams. Thanks to my brother who always leads me in the path and direction of my life. Lastly, I would like to thank my husband who always made me feel loved and cared for. Thanks for his countless round trips, virtual company both day and night, and inexhaustible support for everything I do.

ABSTRACT

As we look toward exascale it is clear that high-capacity HPC storage systems will incorporate the large populations of hard disk drives that have previously only been deployed at cloud-level service providers. Further, with the rapid increase in network performance, the number of disks per storage server will need to be dramatically increased to efficiently pair with current networking technology. With the massive populations of disks integrated within local systems, the probability of various correlated failures across a large number of components becomes a critical concern in preventing data loss. To guarantee the data survival under catastrophic failures, this dissertation emphasizes higher fault tolerance to improve system reliability, provides more flexibility to understand system reliability, and further improves system reliability with less storage overhead.

The first part of this dissertation strengthens existing data protection schemes with higher fault tolerance. We present a novel declustered parity, single-overlap declustered parity (SODP), that ensures at most one overlapping disk between any two stripesets. This maximizes the number of simultaneous disk failures tolerated and minimizes disk rebuild time by balancing parity stripes across disks. Rather than making a trade-off between fault tolerance and rebuild performance, SODP takes the first step to achieve both high fault tolerance and rebuild performance. Our evaluation shows that when compared to the state of the art, SODP can achieve 30x improvements in the probability of data loss during failure bursts.

The second part of this dissertation provides the flexibility to understand how the interactions between fault tolerance and rebuild performance together impact system reliability. We design a practical and flexible tool, fractional-overlap declustered parity (FODP), to explore the trade-offs between the number of failure domains and rebuild performance. This gives us a fine-grained control to accommodate different reliability requirements and system sizes. Furthermore, we introduce FODP-Plus-One to add additional parities on top of FODP to further protect data. Our detailed analysis shows that FODP and FODP-Plus-One yield significant reduction in the probability and magnitude of data loss in the presence of various failure regimes.

The third part of this dissertation is to further explore how SODP/FODP can be integrated into tiered parity, which layers two levels of protection schemes on top of one another. With the tiered architecture, few established principles exist to guide system designs to tolerate both temporal and spatial correlated failures. This work systematically explores the design space for balancing fault tolerance and rebuild performance at each tier and evaluates how different data protection techniques impact the system reliability under various failures regimes. Based on the analysis, we identify a set of design principles that storage architects can use to tolerate correlated failures. By applying these principles, we present a novel tiered parity scheme, Tiered FODP (TFODP), where the top tier is deployed with the minimal FODP technique for high fault tolerance and the bottom tier is designed with the maximal FODP to provide high rebuild performance. Our evaluation shows that TFODP can achieve higher system reliability with less storage overhead.

CHAPTER 1

INTRODUCTION

1.1 Introduction

Modern storage system designs at HPC data centers have traditionally followed two distinct paths: platform-local file systems or platform-shared file systems. The platform-local file system strategy, used by Los Alamos National Laboratory (LANL), focuses on designing a parallel file system to meet the I/O performance and capacity requirements of a single computing platform. The platform-shared design strategy, exemplified by Oak Ridge National Laboratory's (ORNL) center-wide parallel file system [81], instead focuses on building a parallel file system that meets the I/O performance and capacity requirements of multiple HPC computing platforms within the data center. To support the massive storage capacity required to provide critical data services to users, both the platform-local and the platform-shared file systems often incorporate hundreds or thousands of storage nodes and tens of thousands of spinning disks.

The platform-local file systems attached to LANL's Trinity Supercomputer [8] are experiencing significant growth in both disk capacities and populations. At this relatively modest number of disk drives (e.g., 17712) and disk capacity (e.g., 8TB), the Trinity file system incorporates multiple drive models, all of which have differing physical characteristics, and presumably, differing failure characteristics. Furthermore, recent announcements of Exabyte class file systems at national laboratories make it clear that the size of disk populations within local file system is likely to increase substantially in the near future [58]. It is apparent that the emergence of extremely denser, larger, and more heterogeneous disk drives make the threat of correlated drive failures [22] too large to ignore. In particular, it introduces failure bursts [30] and cascading failures [62, 64] that trigger the loss of multiple disk drives within a compressed time window. In addition, common environmental factors (e.g., earthquake) and supporting hardwares (e.g., cable) can cause multiple, simultaneous disk failures to happen within the same physical location. While these correlated failures in time

and space that can cause storage systems to beless reliable have existed for a long time, the modern data protection schemes and the mean time to data loss (MTTDL) [35] calculations still assume that drive failures are independent and identically distributed [49]. This makes the reliability of storage systems more precarious while the data has become more important than ever. To solve these practical issues, this dissertation identifies a set of new principles to tolerate correlated failures and develops a set of new data protection schemes, single-overlap declustered parity (SODP) and fractional-overlap declustered parity (FODP), to prevent data loss.

The platform-shared file systems require larger storage capacities to support big data analysis like scientific discovery, business intelligence, and social media. Disk drive vendors are increasing the capacity of common disk drives with 16TB and 20TB drives commonplace and 24TB drives announced for 2021. At the same time, disk enclosures have also increased in size and now commonly house 104 or 106 drives in only 4U of rack space with current racks typically 42U or 48U tall. Thus, within a single data center rack it has become commonplace to provide greater than 20PB of data capacity provided by more than 1000 disk drives and data centers may host hundreds or thousands of such racks to support modern digital data.

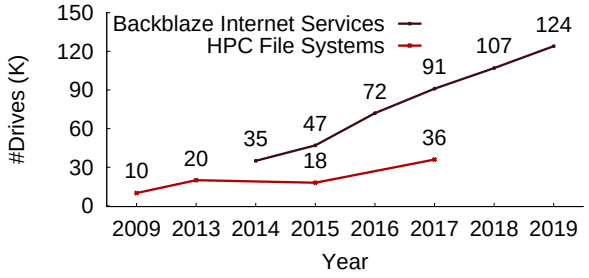

Figure 1.1: **Trends in disk population.** *Trends in the number of disks incorporated into large platform-local and platform-shared file systems at multiple national laboratories versus the number of disks incorporated into the Backblaze cloud service.*

Figure 1.1 shows how storage systems at both cloud archival services and HPC data centers at national laboratories have grown to include tens and even hundreds of thousands of disk drives over the past 10 years. At such unprecedented growth in disk drives, the disk failures [60, 55, 68, 75] are becoming more prevalent in storage systems. Meanwhile, the platform-shared file

systems are required to provide further protection from rack failures [94] and power and cooling failures [47]. Existing protections (e.g., rack-aware placement) against traditional failures domains (e.g., power loss, rack/switch failures, etc) were sufficient to protect against the common types of spatial correlated failures in cloud infrastructure [82, 33, 85, 20] and in on-premise data centers, but the penalty of the network repair traffic is dramatic. For example, Facebook [74] reported 2PB/day network traffic in a 3000-node cluster. To solve this problem, storage architects layer two protection schemes on top of one another to construct tiered parity. Unlike existing schemes [57, 79, 40] that spread data across the distributed system, tiered parity enables most rebuilds of a failed drive to take place entirely locally within each rack and survive the catastrophic failure of an entire rack by using the distributed parity. However, few established principles exist to guide tiered parity designs. This dissertation performs a detailed analysis of tiered parity tradeoffs to quantify the degree to which tiering provides greater fault tolerance and which tier provides higher rebuild performance. By identifying and applying the design principles, we propose Tiered FODP (TFODP) to tolerate various correlated failures [90, 92, 67] with less storage overhead.

1.2 Contributions

The increasing number of disk drives leads one to question whether disk failures could ever be independent and identically distributed – common assumptions leveraged by both data protection schemes and *mean time to data loss (MTTDL)* calculations. As a matter of fact, large populations of disk drives sourced from a small number of manufacturing batches, drives supporting a single consistent workload over time, or even simply large sets of drives that are energized at the same time show the potential to fail in concert. How general-purpose data protection schemes can be used to better tolerate correlated failures is still unclear. This dissertation provides a series of reliability analysis for existing approaches and proposes a set of novel placement schemes that re-explore how to design data protection for modern storage systems where massive failures are possible. The specific contributions of this dissertation are described below.

1.2.1 Correlated Failures

An important contribution of this dissertation is to present a series of correlated failure models that model failures arriving closely in time.

Dense Failures Some recent work [15] in high performance computing systems shows that failures are highly correlated in time resulting in time periods with higher failure density. It observes that 75% of failures just occur within 25% of the system lifetime. As a result, the periods of higher failure density could result in an Mean Time Between Failures (MTBF) [78, 76] multiples higher than the average. In particular, failures in [90, 30] are sometimes time-correlated across hours and days due to environmental effects, firmware bugs, or transient workloads. For example, MarFS storage system [52] reported 432 disk failures that occurred within 24 hours, and within a single disk enclosure in LANL's Trinity file system, 5 drive failures happened in less than 5 days. Although these correlations do not hold over the life of systems, the existence of these highly correlated failures within compressed time windows may make existing storage system data protection schemes highly vulnerable to data loss.

Batch Failures The assumption that failures occur separately from each other is not always valid because many failure types can be traced to batches of components (e.g. a run of disks manufactured with a less effective bearing lubricant). Large storage systems will likely include multiple batches of manufactured components at initial deployment, and the heterogeneity will increase over the life of the system as components are replaced. Once a disk has failed in one batch, it is more likely to trigger another disk failure in that same batch [64]. The reason behind it is both drives come from the same manufacturing batch and share the same fabrication defect. For example, disk with transient defects that may be ignored during the manufacturing process are more likely to fail early in their lifetimes. To study these correlated batch failures, [13, 14] model the initial failures that happen randomly in an independent way while adding a cascading failure characteristic that results in subsequent failures that happen in rapid succession.

Type	Model
Dense failures	Exp($\frac{1}{MTBF}$) || Poisson($\frac{1}{MTBF}$)
Batch failures	Exp($\frac{1}{MTBF}$) & Exp($\frac{1}{0.1*MTBF}$)

Table 1.1: **Correlated failures.** *Poisson and Exponential distribution models used in correlated failures, where MBFT represents Mean Time Between Failures.*

Table 1.1 summarizes the failure models we will use in this dissertation. For dense failures, we use two separate distributions with failure events arriving according to a Poisson distribution or Exponential distribution. For batch failures, we use a model combining two Exponential distributions, where first failures may trigger an additional failure stream that occurs at a 10x faster failure rate [22] and generates 3 to 6 cascading failures [13].

1.2.2 Single-Overlap Declustered Parity

Massive storage systems such as those used for cloud archival services [16, 61] and the file systems at high-performance computing (HPC) data centers [63, 58] provide critical data services to users. These systems are designed with the belief that existing protections against traditional failure domains (e.g., power loss, switch failures) are sufficient to protect against the common types of correlated failures. However, denser and larger disk drives make these storage systems at greater risk for catastrophic failures and data loss [60]. These systems heavily rely on RAID technology using declustered parity, to provide fault tolerance and prevent the loss of valuable data – however, declustered parity schemes were not designed to tolerate large numbers of failures in short windows of time. An important contribution of this dissertation is to reconsider the efficiency of declustered parity schemes and demonstrate that declustered parity suffers from reduced reliability during frequent failures – the type of correlated failures that occur when data centers have to deploy tens of thousands of disks into their data center routinely. To better protect against correlated failures, we identified two additional parity declustering design principles that emphasize data survivability.

- Maximizing the number of simultaneous disk failures tolerated without increasing parity

overhead, and

- Minimizing disk rebuild time by balancing parity stripes across all disks.

This work focuses our attention to emphasize fault tolerance and tolerating correlated failures in declustered parity schemes while maintaining identical rebuild performance. It develops a new data placement scheme called single-overlap declustered parity (SODP), whose basic idea is to have at most one overlapping disk between any two stripesets, to greatly reduce the number of failure domains in declustered parity. To make SODP generally useful we provide two algorithms, optimal SODP (O-SODP) and greedy SODP (G-SODP), for creating SODP strategies for varying the numbers of data blocks, parity blocks, and disks. This work identifies new research opportunities for protecting data at scale. To evaluate SODP, we build an event-driven simulation package, SOL-Sim, for modeling disk failures and rebuilds, and data distributions and reconstructions within Trinity storage system. This work shows that placement designs that follow the SODP principles can dramatically reduce the probability of data loss in the presence of correlated failures.

1.2.3 Fractional-Overlap Declustered Parity

In order to make data services continuously available for both ingest and analysis it has become necessary to employ data reliability schemes that protect against multiple types and sources of failures. Many existing reliability schemes work on two extremes, either enhancing the rebuild performance [45, 38, 77] or improving fault tolerance [65, 25], but how the interaction between fault tolerance and rebuild performance together impact system reliability is still unclear.

While existing copyset approaches [26, 25, 24] was able to explore that tradeoff space for data replication, the copyset technique in Figure 1.2 that guarantees that more than 90% of the total copysets have at most one overlapping node is based on the assumption of large-scale clusters composed of thousands of nodes. However, the large number of configurations that satisfy the copyset constraint makes it typically impossible to apply the copyset approach into the parity groups where the number of data blocks and parity blocks is much larger (e.g. 10 data blocks with

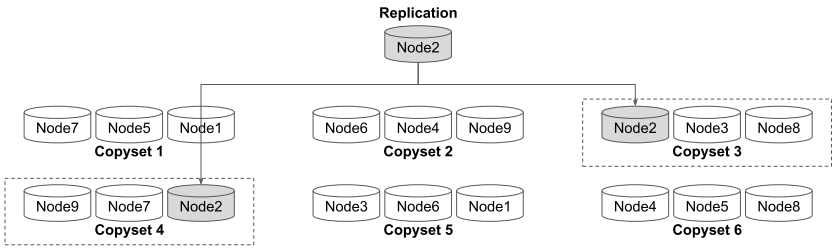

Figure 1.2: **Illustration of the copyset layout technique.** *Each permutation is able to generate independent copysets, and copysets from two permutations are assumed to have at most one overlapping node in the large storage system.*

2 parity blocks) and the number of disks in a typical server is much smaller (e.g. 100 - 200 disks is common). To solve this problem, this dissertation designs a practical tool, fractional-overlap declustered parity (FODP), to flexibly construct the set of parity groups that satisfy the single-overlap disk constraint [53] and explore the trade-offs between the number of failure domains and rebuild performance. This gives us fine-grained control to accommodate different reliability requirements and system sizes. FODP utilizes Mutually Orthogonal Latin Squares (MOLS) [10] to uniformly distribute data and parity blocks across disks and map the given logical units in the specified physical disks. This allows for adding additional parities on top of the FODP data layout, that's what we call FODP-Plus-One, to greatly reduce the granularity of data loss. To the best of our knowledge, this is the first work to achieve the goal of reducing the probability of data loss in the presence of correlated failures while significantly reducing the magnitude of lost data.

1.2.4 Tiered Parity Myths and Facts

While novel data protection schemes, such as SODP and FODP, are able to protect against correlated failures, the recent advances in distributed data layout schemes [25, 46] and distributed data rebuilds [43] require to tolerate spatial correlated failures and are expected to satisfy the 5 9's of reliability required by modern service level agreements [1], but how to combine these advanced techniques including SODP and FODP are is unclear for the storage architect. In general we expect storage system architects to know important parameters such as the amount of data protection that can be reasonably purchased by their organization (e.g. 3-way replication vs 20% overhead

erasure coding). This raises an important question how the given storage overhead brings the highest system reliability. To answer this question, we start with the flat parity (e.g., a single tier) to understand system reliability in terms of temporal correlated failures [23], and then conduct detailed analysis to understand how adding tiers of parity protects data in terms of spatial correlated failures. Normally, storage architects layer two protection schemes on top of one another to construct tiered parity. A common configuration for this approach is a single data protection scheme used within storage system racks that protect against local drive failures and an additional data protection scheme that is implemented across storage system racks and protects against the failure of entire racks of the storage system. To understand whether tiered parity or flat parity is efficient in tolerating correlated failures, we compare the system reliability under various failure regimes to understand when we should add an additional tier. While the tiered parity schemes enable storage architects to design systems tolerant of massive numbers of drive failures, few established principles exist to guide these designs. Through the detailed analysis, this work identifies two principles that emphasize both fault tolerance and rebuild performance in the tiered parity.

- The bottom tier favors larger overlap fractions for higher rebuild performance.
- The top tier uses smaller overlap fractions for higher fault tolerance.

By applying the above two principles we present Tiered FODP (FODP), where the bottom tier is able to achieve faster rebuild time for single disk failure and less rebuild traffics for local data loss, and the top tier is able to provide high fault tolerance with much fewer storage overheads.

1.3 Thesis Statement

We propose a set of novel placement schemes that re-explore how to design data protection schemes for modern storage systems where various correlated failures are possible.

1.3.1 Roadmap

The dissertation outline is as follows:

- Chapter 2 provides related work and background information about different types of erasure codes, which greatly improve data availability and system reliability. On top of that, we review two widely used distribution schemes, RAID and declustered parity (DP), in erasure-coded storage systems.

- Chapter 4 reconsiders the efficiency of traditional declustered parity data protection schemes in the presence of correlated failures and introduces single-overlap declustered parity (SODP) to maximize the number of simultaneous disk failures tolerated without increasing parity overhead and minimizes disk rebuild time by balancing parity stripes across disks.

- Chapter 5 studies the interactions between fault tolerance and rebuild performance and develops a practical and flexible tool, called fraction-overlap declustered parity (FODP), that explores the trade-offs between fault tolerance and rebuild performance. To avoid the loss of large amounts of data in each incident, we further propose FODP-Plus-one to add an additional layer of parity on top of FODP data layout.

- Chapter 6 explores how to integrate SODP/FODP into distributed storage systems that leverage two levels of erasure codes. It performs a detailed analysis of tiered parity to identify the design principles, which guide us to present Tiered FODP (TFODP) for providing higher fault tolerance and higher rebuild performance with less storage overhead.

- Chapter 7 concludes this dissertation and then discusses a couple of related findings and contributions to the storage system designs, which brings up to some future prospects and directions.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this chapter we review the related work and background in erasure codes (Section 2.1), RAID systems (Section 2.2), and declustered parity schemes (Section 2.3).

2.1 Erasure Codes

In this section, we present a brief review of erasure codes [69] that provide storage efficient data redundancy to protect against disk failures and have been widely used in storage systems. Rather than copying data blocks like replication [95, 86], erasure coding (EC) [44, 42, 88, 70, 71, 7] is a way of data protection in which data is divided into k fixed-size data blocks, which are encoded to generate m parity blocks with redundancy information. These k data and m parity blocks form a $stripe$ to tolerate up to m block failures because any surviving k blocks can be decoded to recover the failed blocks. Typically, erasure codes can be categorized into two types: (1) Reed Solomon codes whose encoding and decoding operations are based on Galois Field, which leads to a high computational complexity; and (2) XOR-based codes which only involve XOR operations in encoding and decoding.

2.1.1 Reed Solomon Codes

Reed-Solomon (RS) code is first proposed by Reed and Solomon in 1960 [73] and has its success in various applications including CD-ROMs, DVD, digital TV, wireless communications, and space communications. Given a k-digit plaintext message, each digit is of r bits, RS encodes and sends $n = k + 2s$ digits to ensure that the original message can be reconstructed in case of a maximum of s corrupted digits. The commonly used parameters are $k = 223, s = 16, n = 255, r = 8$. Reed-Solomon (RS) code treats the k-digit plaintext message $(m_0, m_1, \cdots, m_{k-1})$ as the coefficients of

a $k-1$th order polynomial:

$$m(x) = m_0 + m_1 x + m_2 x^2 + \cdots + m_{k_1} x^{k-1}, \tag{2.1}$$

and encodes the k coefficients by sampling n distinct points $(x_i, y_i), i = 0, \cdots, n-1$ on the polynomial:

$$\begin{aligned} m_0 + m_1 x_0 + m_2 x_0^2 + \cdots + m_{k_1} x_0^{k-1} &= y_0, \\ m_0 + m_1 x_1 + m_2 x_1^2 + \cdots + m_{k_1} x_1^{k-1} &= y_1, \\ &\vdots \\ m_0 + m_1 x_{n-1} + m_2 x_{n-1}^2 + \cdots + m_{k_1} x_{n-1}^{k-1} &= y_{n-1}. \end{aligned} \tag{2.2}$$

The encoded message is $(y_0, y_1, \cdots, y_{n-1})$ and the distinct values of $\{x_i\}, i = 0, \cdots, n-1$ in equation 2.3 are agreed upon before sending the message. Given the received message (y_0, y_1, $\cdots$, y_{n-1}) with at most s errors, the first step of decoding is to find a subset of $k+s$ points from $(x_i, y_i), i = 0, \cdots, n-1$ such that a degree $k-1$ polynomial passes through the $k+s$ points. Since the received message has at most s errors, such a subset always exists. Within such a subset, we can find at least k points $(x_{i_j}, y_{i_j}), j = 0, \cdots, k-1$ that are not corrupted since at most s points of the $k+s$ points are in error. Substitute the k correct points into equation 2.3 we have

$$\begin{aligned} m_0 + m_1 x_{i_0} + m_2 x_{i_0}^2 + \cdots + m_{k_1} x_{i_0}^{k-1} &= y_{i_0}, \\ m_0 + m_1 x_{i_1} + m_2 x_{i_1}^2 + \cdots + m_{k_1} x_{i_1}^{k-1} &= y_{i_1}, \\ &\vdots \\ m_0 + m_1 x_{i_{k-1}} + m_2 x_{i_{k-1}}^2 + \cdots + m_{k_1} x_{i_{k-1}}^{k-1} &= y_{i_{k-1}}. \end{aligned} \tag{2.3}$$

In the above we have k equations with k unknowns $(m_0, m_1, \cdots, m_{k-1})$. Solving the above linear systems we can reconstruct the coefficients of the polynomial 2.1 thus recover the original

message. It should be noted that the arithmetic operations $+, \times$ in the above are not real numbers arithmetic operations since the digits are represented by finite number of bits (e.g. 8 bits) and thus are operated using Galois Field arithmetic. In real implementation of Reed-Solomon (RS) code, instead of sampling $n = k + 2s$ distinct points, only $2s$ distinct points are sampled on the polynomial, the coefficients $(m_0, m_1, \cdots, m_{k-1})$ are directly used so the encoded message looks like $(m_0, m_1, \cdots, m_{k-1}, y_0, y_1, \cdots, y_{2s-1})$ [17]. The previous discussion still holds. Once we have found the k correct digits, assuming $k - l$ are the coefficients and $l \leq 2s$ are the sampled points, then only l coefficients are unknown. The l sampled points give l linear equations which solve the l unknown coefficients similarly as equation 2.4.

2.1.2 XOR-based Codes

Though the encoding of RS is relatively straightforward, the decoding is time-consuming which had limited its application in high-bandwidth data delivering. To avoid the high computational cost of Galois Field arithmetic of the Reed-Solomon (RS) code, array codes that solely based on exclusive-or (XOR) operations were proposed in the 1990s [18].

EVENODD code [18] was proposed in 1995 and works with $k + 2$ disks where the data are stored in the first k disks and the parities are stored in the last $m = 2$ disks. Note that k should be a prime number. Consider a $(k-1) \times (k+2)$ array, assume $d_{ij}, 0 \leq i \leq k-2, 0 \leq j \leq k+1$ the ith data in the j_{th} disk. The first parities (in the k_{th} column of the array) is computed by:

$$d_{i,k} = \bigoplus_{t=0}^{k-1} d_{i,t}, \tag{2.4}$$

where $\bigoplus$ represents the exclusive-or (XOR) operation. The second parities (in the $k+1_{th}$ column of the array) is computed by:

$$d_{i,k+1} = S \bigoplus \left(\bigoplus_{t=0}^{k-1} d_{(i-t)\%k,t} \right), \tag{2.5}$$

where $\%$ means the mod operation and

$$S = \bigoplus_{t=1}^{k-1} d_{k-1-t,t}. \tag{2.6}$$

The EVENODD code can tolerate up to two errors. The encoding of the parities are done through XOR operations of the data members as shown in equations 2.4 and 2.5. The decoding is done through XOR operations in an inverse manner.

RDP Code [28] was proposed in 2004 and works with $k+1$ disks where the data are stored in the first k disks and the parity is stored in the last $m = 1$ disk. Note that k is a prime number. Consider a $(k-1) \times (k+1)$ array, assume $d_{ij}, 0 \leq i \leq k-2, 0 \leq j \leq k$ the i_{th} data in the j_{th} disk. Similar to the EVENODD code, RDP uses the diagonal data blocks 2.5 to calculate the parity in the last column. It achieves the optimal computational complexity by directly conducting XOR operations on the diagonal data blocks instead of introducing an extra intermediate parameter. The decoding is done by XOR operations in an inverse manner.

X-Code [89] was proposed in 1999 with the maximum distance separable property. It is an array code of size $n \times n$, where n is a prime number. The data are stored in the first $n-2$ rows. The parities are stored in the last two rows. The parities are calculated from the data in the array along several parity check lines or diagonal lines of certain slopes with addition operation. Note that each column contains both the data and parities. If a data or a parity is in error, then the entire column is considered to be in error state. The X-Code can tolerate up to two corrupted columns

and recover with the remaining $n-2$ columns. The X-code has a minimum column distance of 3 (tolerate arbitrary two corrupted columns). The simple geometrical construction of X-code makes it achieves encoding/decoding optimal complexity.

Low Density Parity Check (LDPC) Code [83] was proposed in early 1960's and become hot topic today due to its fast encoding and decoding algorithms and the ability to recover the original message with large amounts of noises. LDPC codes are linear non-MDS (minimum distance separable) codes defined by sparse bipartite graphs. Assume we have a graph with n left nodes (message nodes) and r right nodes (check nodes). Assume $c_0, c_1, \cdots, c_{n-1}$ the messages associated with the left nodes. The graph gives a linear code $c = (c_0, c_1, \cdots, c_{n-1})$ of length n such that for each check node, the sum of the messages of its neighboring message nodes is zero:

$$\sum_{j \in N(i)} c_j = 0, i = 0, 1, \cdots, r-1, \tag{2.7}$$

where $N(i)$ means the neighboring (message) nodes of check node i. Assume H is the adjacency matrix between the message nodes and check nodes whose size is $r \times n$ with the entry in its ith row and jth column $H_{ij} = 1$ if and only if the ith check node is connected with the jth message node, the relationship in equation 2.7 can be represented in matrix product as $Hc = 0$, where c is the vector of messages and is called a low-density parity-check (LDPC) code. Note that not every binary linear code can be represented by a sparse bipartite graph, only the code has such a representation is called a low-density parity-check (LDPC) code. A LDPC code can be decoded by the Belief Propagation algorithm [32]. The sparse property of the graph allows efficient decoding of the LDPC codes.

Minimal Regenerating (MBR) Code [29] was proposed in 2010. It is designed to decrease the recovery cost with high storage efficiency. In distributed storage systems, large data transfers across the network are common due to the reason that redundancies are continually refreshed as

nodes fail or being replaced. Assume we have a data of size M divided into n blocks (e.g. through erasure codes), the repair bandwidth for a block (disk) of size M/n is M. In literature, the n factor overhead in repair bandwidth is commonly regarded as unavoidable cost along with the benefits of coding. Each storage node of MBR is allowed to store slightly more than M/n bits, which leads to significantly reduced repair bandwidth. In a word, MBR achieves optimal tradeoff between storage overheads and the repair bandwidth.

2.2 Redundant Arrays of Inexpensive Disks

In this section, we introduce the Redundant Arrays of Inexpensive Disks (RAID) [66], a technique to combine multiple disk drives as a whole to improve the capacity, performance, and reliability.

- *Large capacity:* multiple disks in RAID translate to multiple disks' capacities for storing data.
- *Fast performance:* using multiple disks in parallel improves the throughput of read and write I/O performance.
- *High reliability:* RAID distributes redundant information across multiple disks, which enable it to operate and serve users' requests in the presence of one or two disk failures.

2.2.1 Basic RAID Levels

RAID systems are categorized into different basic levels, each of them is a different trade-off between capacity, performance, and reliability. We will discuss six important RAID levels, RAID-0, RAID-1, RAID-3, RAID-4, RAID-5, and RAID-6 in Figure 2.1.

RAID Level 0: Striping RAID level 0 is to stripe user data over multiple disks without redundancy information. With this design, the I/O performance is multiplied by the number of disks. Single large requests can be serviced by multiple disks acting in coordination. and multiple small

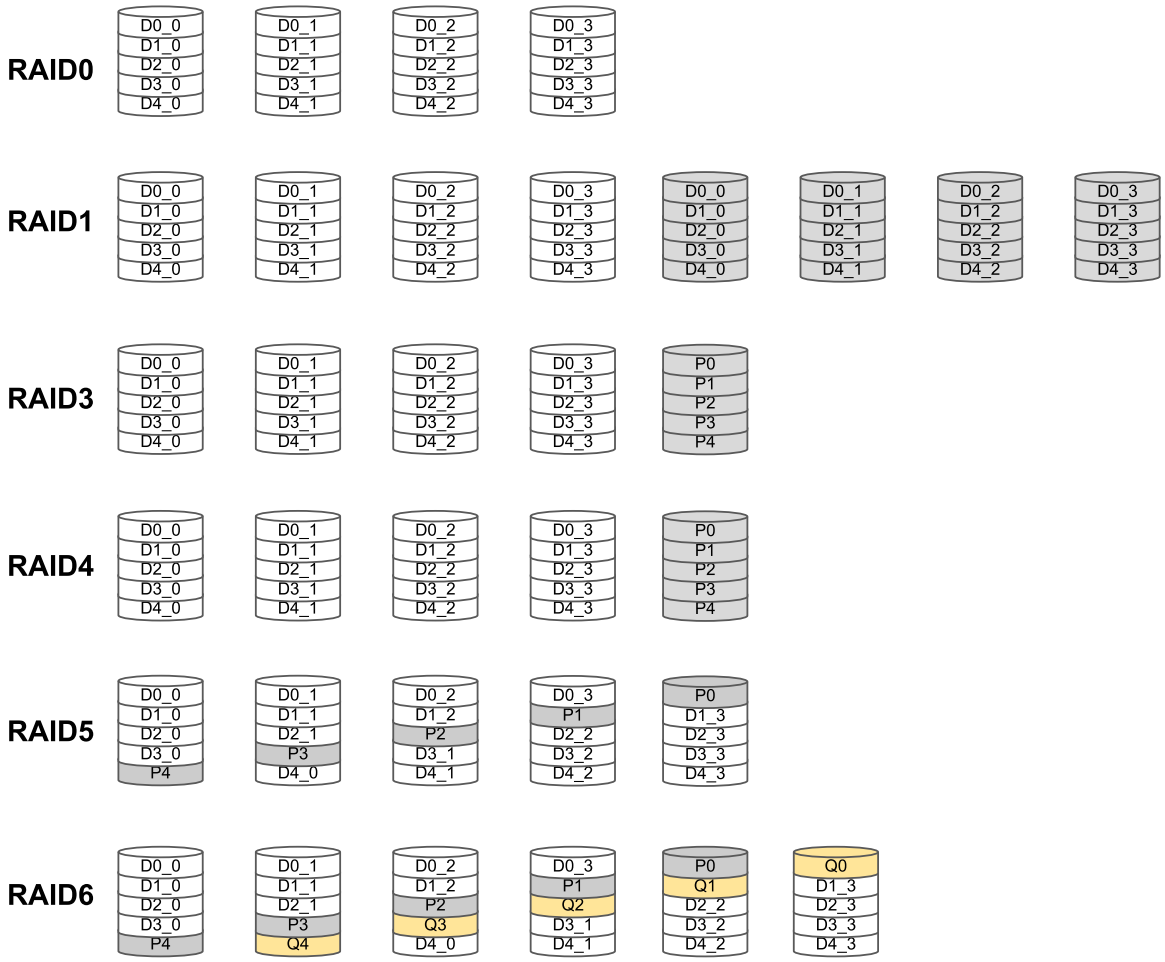

Figure 2.1: **Basic RAID levels.** *This figure compares six important RAID Levels, RAID-0, RAID-1, RAID-3, RAID-4, RAID-5, and RAID-6.*

requests can be serviced in parallel by separate disks. The more disks in the disk array contribute to higher I/O performance, but the more disks lower the overall reliability of the disk array due to the higher chance to experience disk failures.

RAID Level 1: Mirroring To tolerate disk failures, RAID level 1 exploits mirroring, which makes a copy of the data and places it on a separate disk. With the mirroring, when one disk fails, another disk with the copy could be used to serve the user requests. This is necessary to tolerate disk failures and allow continuous operation without data loss. Compared to RAID level 0, data mirroring uses twice as many disks to save the same amount of data. In terms of the read and write

performance, any read request can be serviced by one of the copies and any write request requires to update both, which limits the write performance.

RAID Level 3: Bit-level Parity RAID level 3 is bit-level striping where data is spread over data disks but parities just reside in a dedicated parity disk. Any data update requires to update the parity on the parity disk, which greatly limits the write performance. In addition, the parity disk cannot participate on reads, resulting in slightly lower read performance.

RAID Level 4: Block-level Parity Similar to RAID-3, the data in RAID 4 are spread over data disks and the parities reside in the single parity disk. The only difference is that RAID-4 use block-wise interleaved data, which improve the read parallelism while the write performance is still bottlenecked by the single parity disk.

RAID Level 5: Block-level Distributed Parity To eliminate the write bottleneck of the parity disk in RAID-3 and RAID-4, the RAID-5 distributes the parity units across disks of the disk array. Figure 2.1 illustrates a left-symmetric RAID-5 redundant disk array, where one parity stripe is a set of four data units over which one parity unit is computed. On one hand, the parity units like P_i for parity stripe i are uniformly spread over the disk array, and on the other hand, the data units like $D_{i,j}$ are mapped to the disks continuously.

RAID Level 6: Block-level Distributed Double Parity Similar to RAID 5, the RAID level 6 is block-level striping but with double distributed parities, which are able to tolerate two disk failures. For this reason, RAID-6 becomes more applicable in practical, especially for high-availability systems.

2.2.2 RAID Comparison

To study the RAID large capacity, fast performance, and high reliability, [6] tabulates the comparison of basic RAID levels in terms of storage efficiency, I/O performance, and fault tolerance. Note that the I/O performance is for four types of I/O requests including the sequential read, sequential write, random read, and random write. We assume that a RAID array contains N disks and each disk can transfer data at S MB/s under sequential workloads and R MB/s under random workloads.

	Sequential Read	Sequential Write	Random Read	Random Write	Storage Efficiency	Fault Tolerance
RAID-0	$N * S$	$N * S$	$N * R$	$N * R$	1	0
RAID-1	$\frac{N}{2} * S$	$\frac{N}{2} * R$	$N * R$	$\frac{N}{2} * R$	$1/2$	$1 - \frac{N}{2}$
RAID-4	$(N-1) * S$	$(N-1) * S$	$(N-1) * R$	$\frac{R}{2}$	$(N-1)/N$	1
RAID-5	$(N-1) * S$	$(N-1) * S$	$N * R$	$\frac{N}{4} * R$	$(N-1)/N$	1

Table 2.1: **Basic RAID levels comparison.** *The comparison of RAID I/O performance, storage efficiency, and fault tolerance*

RAID-0 (striping) gets the full bandwidth of the disk array for sequential workloads, therefore the throughput equals to N (the number of disks) multiplied by S (sequential bandwidth of a single disk) for any sequential reads/writes and $N * R$ for any random reads/writes. From the perspective of storage efficiency, RAID-0 can have the full capacities of N disks without any redundant information, which implies zero fault tolerance.

RAID-1 (mirroring) only obtains half of the disk array capacity and tolerates the failure of one disk. When writing out to the RAID-1, each write request requires two disk writes. As a result, the maximum available bandwidth for writing is half of the peak bandwidth (e.g., $\frac{N}{2} * S$ MB/s for sequential writes and $\frac{N}{2} * R$ MB/s for random writes). For sequential read requests, although they are utilizing all disks, each disk receives a request for every other block. This makes each disk only deliver half of the bandwidth and the sequential read in the whole array will only obtain a bandwidth of $\frac{N}{2} * S$ MB/s. What different is that random reads can be distributed across all the disks, which can obtain the full bandwidth (e.g., $N * R$ MB/s).

RAID-4 (parity) uses a dedicated disk for parity information, which results in $\frac{1}{N}$ storage overheads and tolerates up to one disk failure. Sequential reads and writes can both utilize all of the disks except for the parity disk. Therefore, the available bandwidth would be $(N-1)*S$ MB/s for the sequential and $(N-1)*R$ MB/s for the random. Likewise, the random reads are spread across the data disks but not the parity disk, then the corresponding performance is $(N-1)*R$ MB/s. The random writes require to update the related parities, which reside in the same parity disk. Therefore, the parity disk would be the bottleneck under random write workloads. Because the parity disk has to perform two I/Os (read old parity and write new parity), the available bandwidth would be $\frac{R}{2}$ MB/s.

RAID-5 (rotating parity) is identical to RAID-4 to tolerate one disk failure and obtain $N-1$ disk capacities. The only difference is that RAID-5 rotates the parity blocks across disk drives. This makes the random reads to utilize the N disks rather than $N-1$. Without being bottlenecked by the parity disk, the random writes allow for parallelism across disks. Then each RAID-5 write generates 4 total I/O operations and the total bandwidth for random writes will be $\frac{N}{4}*R$ MB/s. For the sequential reads and writes, RAID-5 works identically with RAID-4 to use the aggregated bandwidths of $N-1$ disks. Therefore, the available bandwidth would be $(N-1)*S$ MB/s for the sequential and $(N-1)*R$ MB/s for the random.

2.3 Parity Declustered Data Layouts

In storage systems that stripe data across multiple RAIDs, data is uniformly distributed across the entire system in the normal mode, but in the failed mode, the failed RAID would suffer from massive repair traffic while other RAIDs would be idle. To address this problem, the declustered parity RAID organization distributes data uniformly over all disks rather than several RAID groups. This is preferable because a disk failure would be reconstructed more quickly by having all surviving disks participating in the reconstruction. In this section, we review the state of the art in declustered parity technology and then compare the difference in terms of the common six desirable criteria.

2.3.1 Declustered Parity Schemes

Declustered parity data protection schemes are schemes that map stripes into disk drives where the number of drives (N) is greater than the number of data blocks (k) and the parity blocks (m). We can then denote a declustered parity scheme as being the tuple (N, k, m). For the Trinity file system the declustered parity scheme is based on 8 data blocks, 2 parity blocks, and 40 total disks for a design of (40, 8, 2). Rather than having a single drive dedicated to immediately replacing a failed drive, the declustered parity configures with a distributed spare [21] reserved across all 40 drives. The way that declustered parity works is to distribute $8 + 2$ stripes across 40 disks, where one disk failure makes the rebuild read and write workloads distributed across the remaining 39 disks. However, the rebuild performance in the case of dedicated spare drive (e.g., RAID) will be limited by the write performance of a single spare drive.

BIBD(v, b, r, k, λ) [36] is a collection of b subsets of k elements over a set of v distinct objects, where each object appears in r subsets and each pair of two objects appears in λ subsets. When BIBD is applied in declustered parity, the essence is to find a data mapping to distribute parity stripes of size k over v disks (e.g., declustered layout), where each disk appears in r disk subsets and each pair of two disks appears in λ disk subsets. Note that the complete block design consists of $\binom{v}{k}$ subsets, where each object appears in exactly $\binom{v-1}{k-1}$ of the subsets.

DATUM [11] improves the data mapping by directly computing disks and offsets without using BIBD table lookup. The key idea is to utilize the complete block design, with a particular ordering of the $\binom{v}{k}$ disk subsets, and compute the disks and offsets through the orderings. One drawback of DATUM is the construct a balanced declustered layout, the disk subsets based on complete block designs were originally too large to be usable.

GridRAID [31] is the declustered parity scheme that is used on LANL's Trinity file system. The basic idea is to divide the stripe data into tiles, each of which is a group of stripes across the disk

array. In each tile, it does the data permutation to make the data, parity and spare space spread. With multiple tiles, it benefits from the distributed reconstruction workload during the recovery.

PRIME [12] is designed for prime values of v to approach the ideal declustered layout by slightly relaxing the maximal parallelism property. Like BIBD, PRIME constructs the declustered layout only for a limited set of configurations.

RELPR [12] is similar to PRIME, but it deviates from the ideal in two ways: maximal parallelism and distributed reconstruction. However, RELRP is applicable to arbitrarily configured disk array size v to achieve approximately balanced declustered layout via on-demand calculation.

PDDL [77] declusters the layout by permuting the disks to spread the parity, spare, and client data units throughput the disk array. Obtaining satisfactory base disk permutations is a challenge which makes PDDL only applicable to limited configurations.

dRAID [45] extends the PDDL[77] work and is designed for the use within the Zettabyte File System (ZFS) [9]. To simplify the generation of base permutation, dRAID randomly generates multiple base permutations.

RAID+ [93] enables RAID construction over large disk enclosure to spread reconstruction workload in a balanced way. RAID+ utilizes the Latin squares to construct a declustered layout. The only problem is that the number of v-order mutually orthogonal Latin squares (MOLS) for general v is still an open problem, it's known to exist when v is a power of a prime number, which is exactly the same as BIBD.

OI-RAID [84] is a two-layer encoding architecture and uses BIBD in the outer layer to achieve a balanced data layout. It spreads the data and parity across BIBD groups to enable group fault tolerance, which implies at least three arbitrary disk failures.

2.3.2 *Declustered Parity Comparison*

A critical aspect in the design of a declustered parity is the quality of the distribution of stripes across the total number of drives [38]. These designs can be simply grouped into two basic approaches: balanced designs and unbalanced designs. Balanced designs ensure that the exact number of stripe sets are stored onto each disk while unbalanced design typically only attempt to ensure that the number of stripe sets per drive are approximately similar. Balanced designs have traditionally been judged by six criteria:

- Single failure correcting, no two units of the same stripe are mapped to the same disk.
- Distributed reconstruction, when a disk fails, the reconstruction workload is evenly distributed across the surviving disks.
- Distributed parity, all disks have the same number of parity units.
- Efficient mapping, the mapping from client data to disk is implementable with low time and space requirements.
- Large write optimization, each parity stripe is aligned across the disks such that a stripe can be written without pre-reading the prior contents of any disk.
- Maximal parallelism, a read of n continuous data units induces parallel access from n disks.

Due to the difficulty in balancing the last two criteria, balanced designs are typically selected from a set of well-curated designs that have been evaluated in previous literature. However, because contemporary storage servers may incorporate very large number of disk drives, many modern software based declustered parity schemes (including the scheme used on Trinity file systems) instead use unbalanced designs where the disks involved in each stripe set can not be calculated rather than using balanced designs with a pre-computed table.

Table 2.2 shows how the above schemes satisfy the six original declustered layout criteria. Early parity declustering designs primarily focused on balancing read parallelism while using small

Schemes	Single Failure Correcting	Distributed Reconstruction	Distributed Parity	Efficient Mapping	Large Write Optimization	Maximal Parallelism
BIBD[36]	✓	✓	✓		✓	M
DATUM[11]	✓	✓	✓	✓	✓	M
GridRAID[31]	✓			✓	✓	L
PRIME[12]	✓	✓	✓	✓	✓	H
REPLR[12]	✓		✓	✓	✓	M
PDDL[77]	✓	✓	✓	✓	✓	M
dRAID[45]	✓			✓	✓	L
RAID+[93]	✓	✓	✓	✓	✓	H
OI-RAID[84]	✓	✓	✓		✓	M

Table 2.2: **Comparison of declustered layouts..** *H represents highly approaching maximal parallelism property, and M and L means medium and low, respectively. m is the maximal disk failures that be tolerated in the declustered layout.*

numbers of fixed layouts. Holland in [38] and Reddy in [21, 72] introduce the use of existing balanced incomplete block designs to achieve greater performance during failure recovery. Alvarez shows that there is no general solution when considering arbitrary disk counts [12]. More modern implementations of declustered parity designs move the focus to being flexible in terms of varying numbers of disks by relaxing the strict balancing of prior designs. These unbalanced incomplete block designs can more efficiently utilize wide ranges of disk counts while retaining almost ideal parallelism under failure conditions. Both GridRAID [31] and dRAID [45] use permutations [77] to generate layouts that are only marginally unbalanced while maintaining maximum rebuild bandwidth. As we can see, all these existing works violate the properties of ideal data layout to some extent. Difficulties balancing these criteria lead many modern software based declustered parity schemes to use approximately balanced designs [45, 31]. Finally, no matter the balanced or unbalanced designs are not designed to tolerate the correlated failures. This dissertation is motivated to provide data survival under correlated failures and take the first step to move towards higher reliable next-generation storage systems, where the threat of correlated failures will be hard to ignore.

CHAPTER 3

EXTREME PROTECTION AGAINST DATA LOSS WITH SINGLE-OVERLAP DECLUSTERED PARITY

3.1 Single Overlap Declustered Parity

Prior work on parity declustering has often relied on known BIBD designs to construct perfectly balanced data layouts that attempt to maximize the six factors shown in Table 2.2. However, as conceived, the six criteria do not seek to emphasize data survivability. To that end, we have identified two additional principles that emphasize data survivability during frequent failures:

- Maximizing the number of simultaneous disk failures tolerated without increasing parity overhead, and
- Minimizing disk rebuild time by balancing parity stripes across all disks.

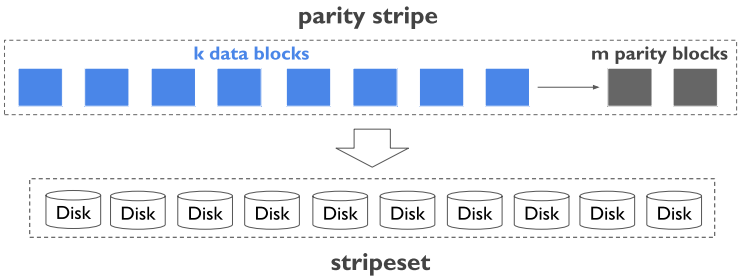

Figure 3.1: **Parity stripe and stripeset.** *Data organized as a parity stripe that will be distributed over a set of disks. A stripeset then is the specific disks selected for a one-to-one mapping with the data and parity blocks.*

In traditional declustered parity, data is encoded into k data blocks and m parity blocks with the $k + m$ blocks forming a **parity stripe**. In practice it is common to use Reed-Soloman codes to construct parity blocks and the notation for a parity scheme is typically shortened to RS(k,m). To satisfy the single failure correcting property, the parity stripe is stored onto a set of $k + m$ disks. In order to tolerate more than m disk failures our techniques require additional care in selecting

how parity stripes are mapped onto disks, and thus we introduce the term **stripeset** to describe a set of disks onto which parity stripes are mapped. Figure 3.1 shows an example of a parity stripe and stripeset. If more than m disk failures occur simultaneously within a single stripeset then the data in this stripeset is lost. In conventional RAID all stripes are located in a single stripeset. With *complete* parity declustering every possible permutation of disks exists as a valid stripeset.

Single-overlap declustered parity, or SODP, is a declustered layout scheme that ensures at most one overlapping disk between any two stripesets. This maximizes the number of disk failures that can be tolerated in a parity scheme with full declustering and maximizes the number of disks participating in a disk rebuild following a disk failure. Figure 3.2 illustrates the layout of the SODP

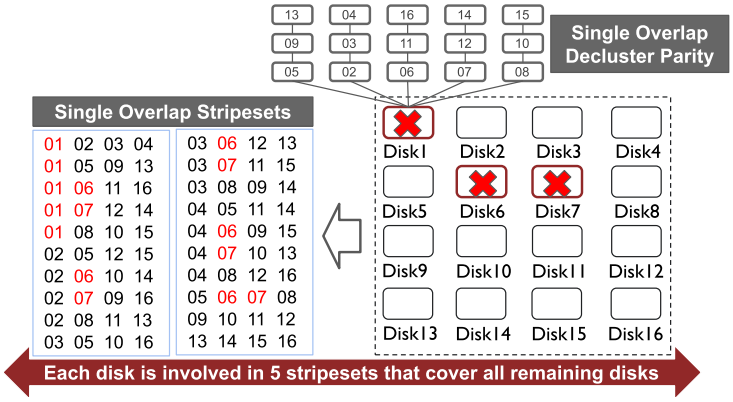

Figure 3.2: **Single overlap declustered parity (SODP).** *A table of the full set of 4-disk single overlap stripesets chosen from a population of 16 total disks. With an RS(2,2) coding we can see that 6 simultaneous drive failures can be tolerated without data loss. The number of failures tolerated within a SODP layout depends on the parity scheme selected.*

design across 16 disks with RS$(2, 2)$ encoding. As shown, only 20 total stripesets are required to construct a fully declustered layout, where every disk participates in a stripeset with every other disk. To provide an example, Disk1 participates in 5 stripesets, but none of the other disks appear more than once in those same stripesets. If Disk1 fails, the remaining 15 disks can be used for recovery, which provides the same rebuild performance as traditional parity declustering. This is true for all 16 disks. Furthermore, we can see that disks 1, 2, 5, 7, 10 and 11 may fail simultaneously and there are no stripesets experiencing 3 failures, and thus no data is lost. Therefore, rather than tolerating only 2 failures, the SODP layout can tolerate 6 failures without experiencing data loss.

3.1.1 *Optimal SODP*

We introduce a stripeset construction algorithm we call Optimal SODP, or O-SODP, that uses matrix manipulation to minimize the number of stripesets. Before presenting the full O-SODP algorithm, we walk through the construction of single overlap stripesets using the above example, where each disk participates in 5 stripesets. First, the 16 disks are organized into a 4x4 disk matrix with rows a, b, c, d and columns $1, 2, 3, 4$ as shown in Figure 3.3. There are three steps to construct the single overlap stripesets:

Generate Row-based Stripesets

Each row (e.g., a, b, c, or d) consists of 4 disks, which form a row-based stripeset.

Generate Column-based Stripesets

Each column (e.g., 1, 2, 3, or 4) also consists of 4 disks, which construct a column-based stripeset.

Generate Row-column Stripesets

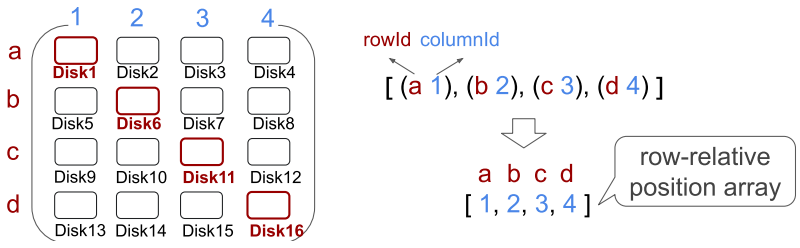

Figure 3.3: **Disk matrix and row-relative position array.** *a, b, c, d represent rows with one-to-one correspondence to the position array [1, 2, 3, 4], which comes from the position coordinate [(a 1), (b 2), (c 3), (d 4)] and aims to choose disks from different rows and columns to create row-column stripesets.*

The key idea of row-column stripesets is to choose 4 disks from different rows and columns. As shown in Figure 3.3, the simplest example is to choose disks on the diagonal, whose positions are denoted as [(a 1), (b 2), (c 3), (d 4)]. We simplify this notation into a *row-relative* position array $[1, 2, 3, 4]$, which represents the row-column stripeset [Disk1, Disk6, Disk11, Disk16].

To generate the remaining row-column stripesets while maintaining a balanced declustered layout we define a new algorithm called **shuffle permutation**. The objective is to swap all possible two position pairs in the diagonal row-relative position array to create new position arrays. As shown in Figure 3.4, if we first swap the position pair $(1, 2)$, it is obvious the next swap should be the pair $(3, 4)$, which generates a new *row-relative* position array $[2, 1, 4, 3]$ that denotes the row-column stripeset [Disk2, Disk5, Disk12, Disk15]. In this extremely simple example we produced three new row-relative position arrays which were permutation shuffled from the initial position array $[1, 2, 3, 4]$. After completing shuffle permutation, the resultant 4 position arrays form a position matrix, which has a unique value for each column. Therefore, the position matrix is able to generate 4 non-overlapping row-column stripesets which cover the entire disk matrix. Algorithm 1 shows the pseudocode for implementing the shuffle permutation algorithm.

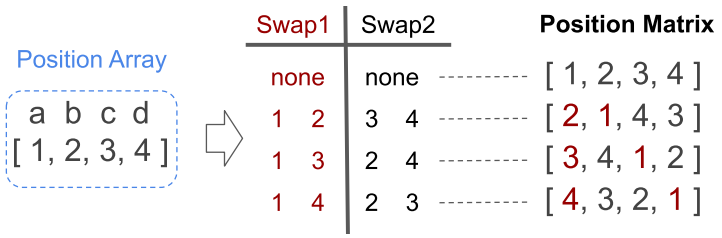

Figure 3.4: **Shuffle permutation.** *Swapping all possible position pairs in the initial position array.*

To generate the additional row-column stripesets containing at most one overlapping disk per stripeset, fix one position in the row (e.g., a) and rotate the other three positions of that row (e.g., b, c, d) as shown in Figure 3.5. A single rotation of the position array $[1, 2, 3, 4]$ leads to a new position array $[1, 3, 4, 2]$. By applying a single rotation to the other position arrays in the matrix, a new position matrix is generated. This newly formed position matrix corresponds to 4 non-overlapped row-column stripesets. The new stripesets are single overlapping with the position matrix from which they were derived. Furthermore, rotating the position array $[1, 2, 3, 4]$ twice leads to another new position array $[1, 4, 2, 3]$. Correspondingly, another new position matrix is formed to generate another 4 new row-column stripesets, all of which satisfy the single overlap property. The process will continue until rotation isn't possible anymore. Therefore, 3 position

Algorithm 1: Shuffle Permutations.

Input: initialPosition ← [1,2,..., c], c columns
Output: P=$\{P_1, P_2, ...\}$, shuffled position arrays
function CREATESHUFFLEARRAYS($B, N, k+m$)
    P = {}
    for *i = 1: c-1* **do**
        for *j = i+1 : c* **do**
            temp = initialPosition
            empty temp[i] and temp[j]
            P' = createShuffleArrays(temp)
            for *k = 1: length(P')* **do**
                tmp = P'_k
                tmp.insert(j) at i_{th} position
                tmp.insert(i) at j_{th} position
                P.add(tmp)
            end
        end
        end
    return P
end function

matrices are available from row-column stripeset generation, or alternatively each disk is included in three row-column stripesets.

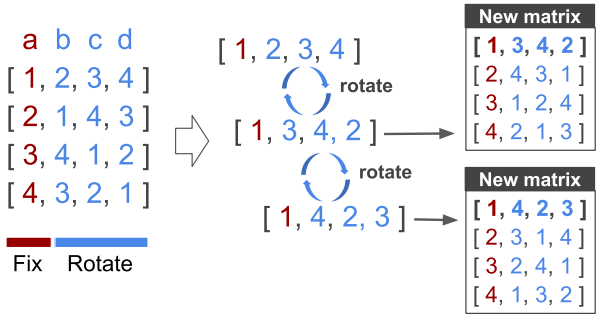

Figure 3.5: **Fix and rotate.** *Fix one position (in red) and rotate the remaining positions (in blue) to generate the new position matrices*

To conclude this example, combining all row-based, column-based and row-column stripesets based on shuffle permutation and rotation, the above example generates 20 stripesets in total across 16 disks with each disk included in exactly 5 stripesets.

CHALLENGES: When the size of a stripeset is large or an odd number, how do we do pair-wise swap in the permutation shuffle? Figure 3.6 illustrates the case of the stripeset size being 7, which leads to the initial diagonal position array $[1, 2, 3, 4, 5, 6, 7]$ with corresponding rows a, b, c, d, e, f, g. To generate a position matrix, we will swap 1 and 2 in the second position array, 1 and 3 in the third position array and so on. Unlike the previous 4-column case where the remaining pair-wise swap is obvious, our new 7-column case leaves the remaining swaps with $\frac{\binom{5}{2}\binom{3}{2}\binom{1}{1}}{2!}$ possibilities.

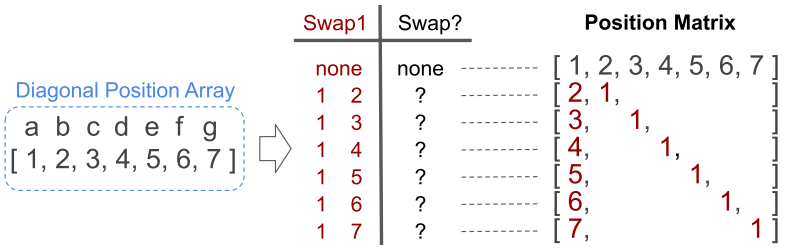

Figure 3.6: **Initial swap of stripeset size 7.** *The corresponding position array is [1,2,3,4,5,6,7] and the corresponding position matrix is to swap positions with the first row.*

To solve the above challenge, we introduce the concept of **rotate distance**, which indicates the clockwise distance between any two positions in the rotate space of the position array. For example, the rotate distance from 3 to 6 is two, because it has to walk through 4 and 5. To satisfy the SODP property, we should guarantee the following constraint:

Constraint #1: Rotate distance before and after swapping cannot be equal.

With the same rotate distance, the new position array will eventually overlap multiple positions with the diagonal position array. As shown in Figure 3.7, if we swap the position pair $(3, 6)$ in the second position array, the rotate distance from 3 to 6 is still two (e.g., walk through X and 1), which is equal to the previous rotate distance. As a result, after rotating the second position array 3 times, it will double overlap with the diagonal position array. To prevent this from occurring, we identify the following property to satisfy constraint #1 for any single position pair (a, b).

$$d_r(a \rightarrow b) \neq d_r(b \rightarrow a)$$

where $d_r(a \rightarrow b)$ represents the rotate distance from a to b. In the example, this will prevent the swapping of position pairs $(3, 6)$ and also $(4, 7)$ in the second position array.

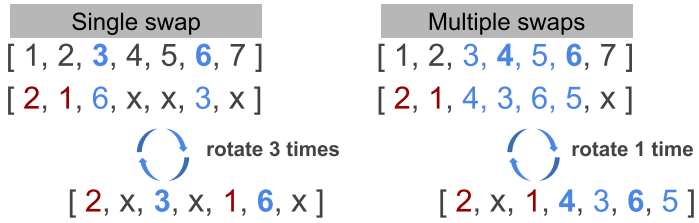

Figure 3.7: **Constraint 1.** *Properties for single and multiple position pairs swapping in permutation shuffle*

Next, if we try to swap both $(3, 4)$ and $(5, 6)$ at the same time, the rotate distances $d_r(3 \rightarrow 5)$ and $d_r(4 \rightarrow 6)$ for the second position array are equivalent to those in the diagonal position array. This means if we were to then rotate the already swapped second position array 1 time, it would still cause an overlap (e.g., 4 and 6) with the diagonal position array. As we can see, both of the single position pair swaps are feasible, but swapping them together creates a conflict. To avoid the multiple overlaps resulting from multiple position pair swaps, we have to guarantee any two pairs (a_1, b_1) and (a_2, b_2) meet the following requirement to satisfy constraint #1.

$$d_r(a_1 \rightarrow b_1) \neq d_r(a_2 \rightarrow b_2)$$

This prevents the swapping of position pairs with the same rotate distance in the diagonal position array. For example, if we swap the position pair $(3, 4)$, it will exclude other position pairs $(5, 6)$ and $(6, 7)$. The only feasible additional swap is the pair $(5, 7)$ and then the remaining position 6 is left untouched. The resultant second position array is $[2, 1, 4, 3, 7, 6, 5]$, which will not overlap more than one position with the diagonal position array regardless of how many rotations are applied.

Now we need to prevent multiple overlaps between the second and subsequent position arrays. The most straightforward way is to avoid swapping the same position pairs. Another approach, utilizing the given second position array, would be to add one along the diagonal based on the circle shown in Figure 3.8. As you can see, the positional elements inserted alongside the diagonal 1s

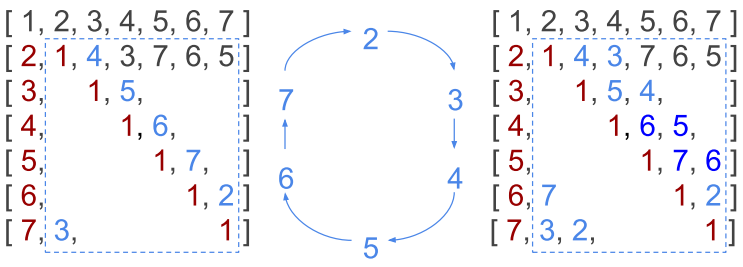

Figure 3.8: **Constraint 2.** *Auto-generation of other position arrays with a given second position array*

are $4, 5, 6, 7, 2, 3$. By applying the same principle, the next set of additional positional elements are $3, 4, 5, 6, 7, 2$. At this point, one can notice the repeating $6, 5$ and $7, 6$, which have already occurred in the second position array. This materializes because in the second position array $d_r(4 \rightarrow 3)$ is equal to $d_r(6 \rightarrow 5)$, and by adding 1 both $6, 5$ and $7, 6$ appear again thus causing a double overlap in the given second position array. To guarantee the derived arrays will never violate the SODP constraint with the second position array, we identify a second constraint:

Constraint #2: Rotate distances in the second position array must be distinct.

This means if we have the successive $4, 3$ in the second position array, it is not allowed to include successive $7, 6$ or $6, 5$. Otherwise, other derived position arrays will experience multiple overlaps with the second position array.

By combining the two constraints above, we are able to create a feasible second position array and a corresponding position matrix. Note that sometimes, a perfectly balanced declustered layout based on our position matrix with the required parameters cannot be found, we publish our feasible position matrices in [41].

THEORETICAL ANALYSIS To demonstrate that O-SODP minimizes the number of stripesets, S, we assume a disk array of size N, where each stripeset consists of $k + m$ disks. To count the disk pairs (i, j), we have

$$S * \frac{(k+m)(k+m-1)}{2} \geq \frac{N(N-1)}{2}$$

which guarantees the disk pairs in stripesets cover all disks pairs in the N-disk array. The size of

S can be formulated as:

$$S \geq \frac{N(N-1)}{(k+m)(k+m-1)}$$

To count the number of pairs (s, d) where s is a stripeset and d is a disk in the stripeset, we have the following equation:

$$S * (k+m) = N * r$$

Here r is the number of stripesets per disk. To count the triples (s, d_1, d_2) where d_1 and d_2 are distinct disks and s is a stripeset that contains both, we have the following equation:

$$1 * (N-1) = r * (k+m-1)$$

where O-SODP makes any pair of disks (e.g., d_1 and d_2) appear in one stripeset. By combining the two equations, the size of S equals $\frac{N(N-1)}{(k+m)(k+m-1)}$, which is the minimum.

Figure 3.9 compares the total number of stripesets using O-SODP with the configurations identified in prior BIBD literature [38]. Additionally, we compare the number of stripesets per disk using O-SODP, being that the number of stripesets per disk directly reflects the rebuild performance. To be specific, the number of surviving disks participating in single disk rebuild is:

$$min\{\textit{stripesets-per-disk} * (k+m-1), N-1\}$$

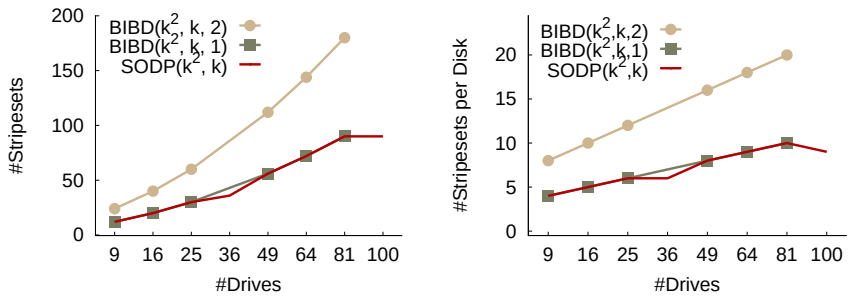

Figure 3.9: **Comparison of BIBD and O-SODP.** *Total number of stripesets and number of stripesets per disk for optimal SODP and defined BIBD configurations*

We see that in general our O-SODP algorithm is able to match the BIBD performance with $\lambda = 1$ while the dips show the G-SODP (see Section 3.1.2) results for configurations not having a known BIBD configuration. We also include the BIBD stripeset counts for the same configuration with $\lambda = 2$ to demonstrate the degree to which higher λ generate additional stripesets which do not further improve rebuild performance but do reduce the total number of disk failures that can be tolerated. If we consider a stripeset as a failure domain we see that these BIBD designs have a greater number of failure domains with a lower degree of fault tolerance. However, O-SODP is not guaranteed to generate a set of single-overlap stripesets for all configurations (even when the stripeset size is smaller than the square root of the number of disks).

3.1.2 Greedy SODP

For arbitrary numbers of disks and arbitrary numbers of data blocks (k) and parity blocks (m) we designed the Greedy SODP algorithm, G-SODP, to achieve *nearly* single overlap declustering. Put simply, G-SODP sacrifices a small amount of rebuild performance to gain a modest improvement in disk failure tolerance. As we will see later in Section 3.2 this tradeoff turns out to be surprisingly effective when we evaluate the probability of data loss under failure bursts. In other words, G-SODP achieves a result very similar to that of O-SODP by slightly reducing the rebuild performance, which in turn can tolerate more disk failures.

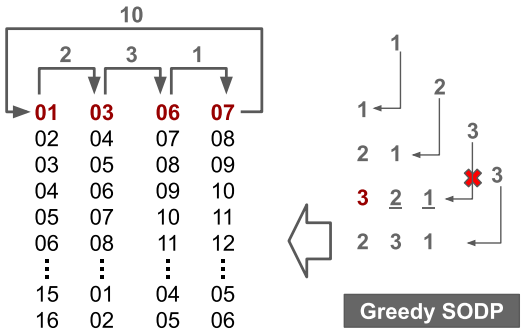

Figure 3.10: **Greedy SODP.** *Base stripeset is [1,3,6,7] and the i_{th} derived stripeset is generated by adding i mod N based on the base one.*

The basic idea of G-SODP is to create one or more base stripesets and derive the i_{th} stripeset

by adding i mod N. Figure 3.10 illustrates how to construct $2 + 2$ stripesets within 16 disks. G-SODP uses $[1, 3, 6, 7]$ as the base stripeset and add i to the disk obtained from the base stripeset, which is able to generate 15 derived stripesets. Utilizing this process, G-SODP guarantees each disk participates in an equal number of stripesets (e.g., 4) to achieve a balanced declustered layout comparable to O-SODP.

Algorithm 2: Greedy SODP (G-SODP).

```
Input: N, disks per server; s, stripe size
Output: B={b_1, b_2, ..., }, base stripesets
while b = createBaseStripesets(B, N, s) do
    B ← b
end
function CREATEBASESTRIPESETS(B, N, s)
    b = [1]
    for i = 2: N-1 do
        for j = 0 : length(b) do
            b' = [b[0:j-1], i, b[j:end]]
            if distances not in B then
                if distances in b' are not equal then
                    b = b'
                    break
                end
            end
        end
        if length(b) == s-1 then
            return b
        end
    end
end function
```

CHALLENGES: The fundamental challenge of G-SODP is to obtain base stripesets, which aim to construct distinct distances between any two elements inside the given base stripeset. In Figure 3.10, the base stripeset $[1, 3, 6, 7]$ is composed of distinct distances 2, 3, 1, and 10. As we can see, any one, two or three accumulative distances are impossible to be equal to any other existing distance, which guarantees adding any i will not yield more than one overlapping disk with any previously generated stripesets. Utilizing these methods, G-SODP is able to generate 16 single

overlap stripesets instead of the 20 generated by O-SODP. Note that, sometimes it's not always feasible to have any x accumulative distances that are different, given this we first ensure that one accumulative distance is different, then the two accumulative distances different and so on, which maximally reduces the multiple overlaps among stripesets.

Algorithm 2 presents the greedy algorithm pseudocode to generate base stripesets. Suppose a base stripeset contains $k + m$ disks, our goal is to select $k + m - 1$ disk distances. We start with the minimum distance 1 and inject it into an empty distance array. For any next distance (e.g., 2 or 3), we can put it before or after the existing distances in the array. The valid injection is to ensure no equal disk distance. For example, Figure 3.10 injects the next distance 3 at the beginning of the array, which leads to an equal distance of the sum of next two distances (e.g., $3 = 2 + 1$). In this case, the derived stripesets based on this base stripeset will cause two disks to be overlapping.

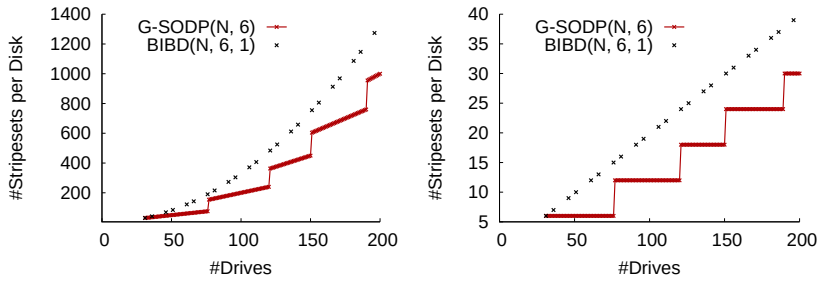

Figure 3.11: **Comparison of BIBD and G-SODP.** *Total number of stripesets and number of stripesets per disk for greedy SODP and a defined BIBD configuration*

Figure 3.11 compares the total number of stripesets and stripesets per disk between BIBD and G-SODP. Here, BIBD(N,6,1) represents a full declustering layout over N disks, which only exists for a limited number of configurations. G-SODP aims to find a balanced layout for arbitrary disk size (e.g., $N > 31$). The comparison results show that G-SODP has fewer stripesets than BIBD(N,6,1), which indicates a gap between G-SODP and the full declustering BIBD. When a disk fails, G-SODP cannot guarantee that every surviving disk participates in that disks recovery (e.g., shorter rebuild time). However, it still attempts to maximize the rebuild performance in a balanced way, which in turn generates less stripesets than O-SODP to tolerate more concurrent disk failures.

COMPARISON OF O-SODP AND G-SODP Both O-SODP and G-SODP are able to generate single overlap stripesets in a balanced way. The only difference is O-SODP constructs perfectly balanced declustered layouts, where each pair-wise set of disks appears in exactly one stripeset. The perfect balance and exact overlap value of one make the number of stripesets minimized in the declustered layout. This design is not possible for all disk configurations (e.g., 8 total disks using 3-disk stripesets). G-SODP relaxes the overlap constraint slightly such that a few pair-wise combinations are not generated but a wider range of disk configurations are supported. Thus G-SODP provides greater configuration flexibility and fault tolerance while sacrificing a small amount of rebuild performance.

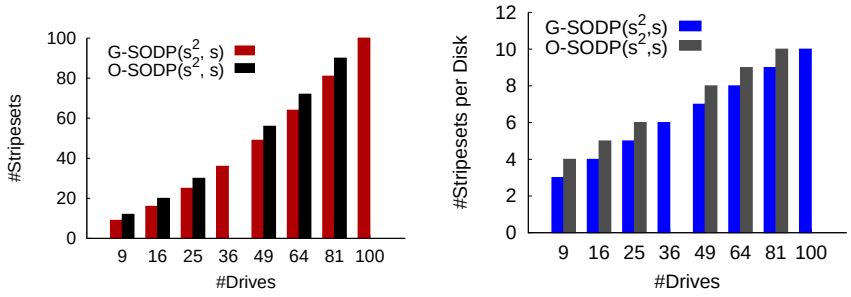

Figure 3.12: **Comparison of O-SODP and G-SODP.** *Total number of stripesets and number of stripesets per disk for O-SODP and G-SODP*

Figure 3.12 compares the total number of stripesets and stripesets per disk between O-SODP and G-SODP. As previously stated, some configurations (e.g., 36 and 100 disks) are not supported in O-SODP, other configurations show similar results between G-SODP and O-SODP. When considering the number of stripesets per disk, G-SODP always generates one less stripeset than O-SODP does accounting for the small difference in total stripesets. In section 3.2, we will provide a detailed comparison of O-SODP and G-SODP protecting against concurrent failures.

3.2 Evaluation

3.2.1 SOL-Sim Design

SOL-Sim is a discrete-event simulator that characterizes the reliability of erasure coded storage systems. Written in Python, SOL-Sim extends SimEDC [94] to supports additional erasure codes, chunk placement schemes, and data re-protection algorithms. Figure 3.13 shows the high-level SOL-Sim architecture which uses failure traces, disk layouts, and data protection schemes as input and returns timings and reliability metrics such as the probability of data loss (PDL) as output. SOL-Sim is designed to simulate reliability over longer periods of time (e.g., 5 years). One key component of the SOL-Sim architecture is the ability to use a complementary tool, CoFaCTOR [53], to evaluate multiple storage system designs over their entire lifetime easily.

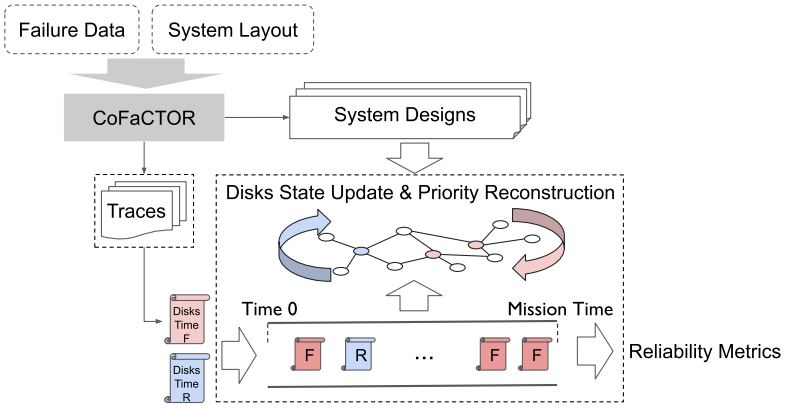

Figure 3.13: **SOL-Sim architecture.** *SOL-Sim consists of failure traces analysis, failures events and repair events handling, system state update, and reliability metrics output.*

SOL-Sim Workflow In order to evaluate the data protection schemes in this paper we have combined the output of CoFaCTOR with simulation to evaluate the probability of data loss over a variety of realistic failure workloads. SOL-Sim stores all events in an event queue, which always returns the event with the smallest timestamp. If the event is a failure event, it will update the disk state, such as the clock and failure status. If the event is a repair event it updates the corresponding disk's clock, repair status (e.g, critical to degraded or degraded to normal), and repair priority.

SOL-Sim and CoFaCTOR provide the following features:

System Layout & System Failure Data In order to generate a large set of realistic failure traces CoFaCTOR is seeded with both a system layout and a set of failure data collected from real system data. The system layout describes physical characteristics of the system that influence failure such as the physical position with the data center (data center row and rack number), the vertical position within the rack and even the disk positions within the storage enclosure (called the drawer row here to differentiate disks near the front of the enclosure, disks in the center of the enclosure, and disks near the rear of the enclosure). Detailed layout data in combination with positional data is critical in generating the set of failure traces and candidate system designs for SOL-Sim.

Failure Traces & System Designs CoFaCTOR generates an arbitrarily large number of synthetic failure traces for use by SOL-Sim. These synthetic failure traces are generated using the survival analysis models seeded with real failure data. The second input into SOL-Sim is the set of system designs output by CoFaCTOR. These configurable system designs enable us to apply the generated failure traces to flexible system designs that explore both the physical system design space and the data protection algorithms used. For example, in this analysis we are able to alter the number of disks per enclosure (i.e. the failure domain for the declustered parity grouping) to explore future storage systems which are expected to be much denser than our existing system design.

Chunk Placement SOL-Sim enables the use of multiple declustered placement algorithms and data protection schemes in conjunction with the failure data including: traditional RAID, complete declustered parity designs, dRAID, and both O-SODP and G-SODP depending on the availability of a single-overlap configuration for that design point.

Priority Reconstruction SOL-Sim also implements a priority reconstruction algorithm that mimics those used in enterprise-grade production storage systems. If multiple drives fail within a server,

to minimize data availability risk, any stripes that are missing two blocks are given priority for reconstruction. This approach is called critical reconstruction. After those critically affected stripes are reconstructed, the rest of the stripes continue to be reconstructed (called degraded reconstruction). While this algorithm is the state of the art it is not widely available for production systems.

3.2.2 Trinity Storage System

LANL's Trinity file system is composed of two identical file systems accessible through the same set of gateway nodes within the Trinity platform. The identical file systems are organized as two parallel aisles of racks within our data center to both enable easier servicing/upgrades and protect against some types of failures external to the file systems. Each file system has 6 total metadata servers and 216 Lustre object storage servers (OSS) each with a single 41 disk Lustre object storage target (OST). OSS node pairs share a single two-drawer 84-bay disk chassis with 41 drives assigned to each of the OSS (the remaining 2 slots contain SSDs used as journal devices). Each drawer within the 84 disk enclosure is composed of 3 rows with 14 drive slots per row. Row 1 holds the 14 drives nearest the front of the drawer and row 3 holds the 13 drives and the SSD at the rear of the drawer. The 41 disk OSTs use a declustered parity approach to construct 8+2 protected stripes with 128KiB stripes forming a 1MiB stripe set. The simulations presented in this paper assume that all drives are 6TB Seagate Makarra drives, the file system is at 60% capacity utilization, and the rebuild disk bandwidth has been set at approximately 50MB/s per disk [31].

3.2.3 Failure Traces Analysis

To create a sufficient number of failure streams to perform our reliability analysis we use the trinity file system configuration, two years of drive failure data, and CoFaCTOR to generate 10000 5-year long failure traces. We note in individual experiments where the system design has been altered to explore alternative storage system designs (e.g. changes in disk capacity, disk enclosure size, or total number of drives). We also use CoFaCTOR to generate 10000 traces where a fixed

percentage of disks fail instantaneously or over a fixed period of time (using Poisson arrivals) to simulate catastrophic failure events such as power outages.

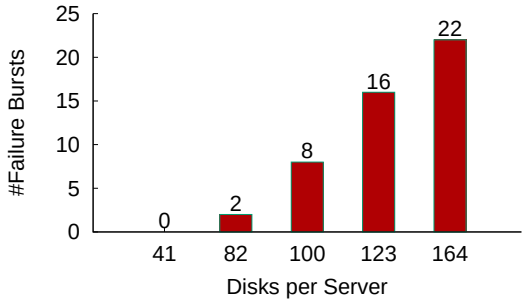

Figure 3.14: **Comparison of failure bursts with varying disks per server.** *The number of 3 disk failures occurring within an 8 hour window using traces generated by CoFaCTOR*

System Lifetime Failure Analysis In order to study the correlated failures, we investigate the number of failure bursts (e.g., multiple failures in a failure domain within 8 hours) with varied numbers of disks per server. As shown in Figure 3.14, among 10000 failure traces, larger disk enclosure sizes are more likely to encounter failure bursts and possibly data loss. Figure 3.14 shows the likelihood of a failure burst as we alter the number of disks allocated to each server. Not surprisingly, as we increase the number of disks per server the number of failure bursts over the life of a storage system increases.

Disks per Server	Scheme	Spares	6TB	14TB	20TB
123	DP	1	0	4	7
		2	0	4	7
		3	0	4	7
	SODP	1	0	0	0
164	DP	1	1	2	5
		2	0	1	4
		3	0	1	4
	SODP	1	1	1	1

Table 3.1: **Data loss events comparison.** *Comparison of the number of 5-year traces with at least one data loss event. We evaluate different disks per server, data protection schemes, and the number of distributed spares for differing disk capacities*

Table 3.1 then shows how well Trinity's declustered parity scheme protects against data loss events compared with an SODP data placement scheme while varying the spare capacity and drive capacity. We use the 123 and 164 disk configurations with G-SODP because an insufficient number of disks exist for an RS(8,2) SODP configuration at the smaller disk counts. We see that SODP prevents data loss with a lower sparing overhead better than the existing Trinity file system, however a rapid burst of failures within a single stripe before copyback completes still causes a single data loss event in the 164 disk configuration.

3.2.4 Catastrophic Failures Analysis

To explore how different data protection schemes perform during common burst failure scenarios (e.g. a power outage or cascading failure that results in a large number of drives failing in a short time window) we simulated failures correlated in time but not correlated in any other dimension. We also compare the corresponding rebuild performance for schemes including traditional RAID, dRAID, Trinity's declustered parity (DP), O-SODP and G-SODP.

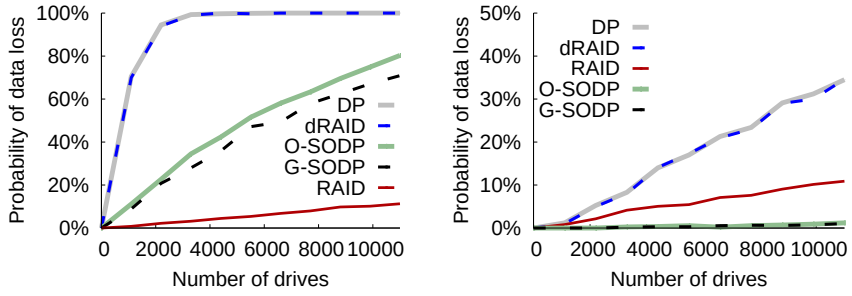

Figure 3.15: **Comparison of the probability of data loss under simultaneous failures and failure burst over 24 hours with varying disk drives in the system.** *The probability of data loss with failure of 1% of the drive population as the number of disk drives are scaled. In the left figure we see that if the drive loss is instantaneous a non-overlapping RAID scheme provides the greatest fault tolerance. However in the right figure we distribute the failures over a 24 hour period which allows the fast rebuild performance of declustered schemes to greatly reduce the overall probability of data loss.*

Effect of Disk Population Size Figure 3.15 shows the probability of data loss (PDL) during a burst failure of 1% disk failures and increased number of disks. In particular, we increase the total number of disks from 1100 to 11,000 and examine the PDL for 1% drive population failure

instantaneously and randomly distributed over 24 hours. In Figure 3.15(a), failures occur in an instantaneous burst and rebuild time is irrelevant. Non-overlapping RAID can tolerate the most simultaneous failures with 11.3% PDL. The greater fault tolerance of the SODP schemes protects data at smaller disk counts but is not sufficient for large disk populations. With Trinity's declustered parity and dRAID we see a 100% chance of data loss over 6600 disks. In Figure 3.15(b), the failures simulate a cascading failure occurring over 24 hours, thus data is rebuilt during the 24-hour failure period for the declustered parity schemes. By tolerating more failures and having fast rebuild performance, G-SODP and O-SODP have 1.05% and 1.2% PDL for 11,000 disks, respectively. Trinity's declustered parity and dRAID also benefit from fast rebuild performance, but the lack of greater fault tolerance lowers the PDL to only 34.6%. Interestingly, even with failures distributed over 24 hours RAID is better than Trinity's declustered parity and dRAID indicating that in this experiment fault tolerance is more important than rebuild performance.

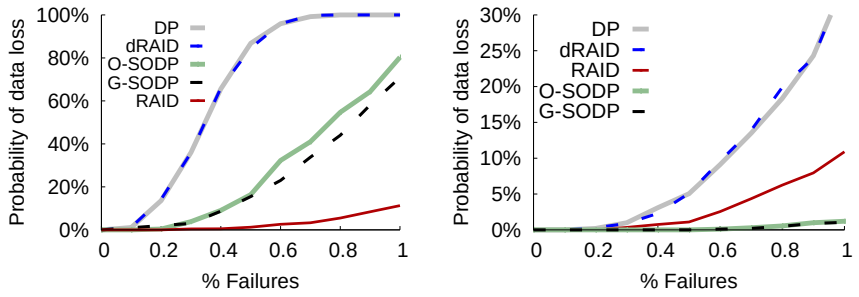

Figure 3.16: **Comparison of the probability of data loss under simultaneous failures and failure burst over 24 hours with varying failure burst sizes.** *The probability of data loss varied with the percentage of failed disks in a population of 11000 drives. During instantaneous failures in the left figure traditional declustered parity schemes experience a 100% chance of data loss once approximately 0.6% of the drives have failed. When the failures are distributed over 24 hours SODP schemes exhibit a less than 1.2% chance of data loss.*

Effect of Burst Size Figure 3.17 varies the number of failed disks in a population of 11,000 drives from 0% to 1% to compare each of the parity placement schemes. As before we examine the probability of data loss (PDL) for simultaneous failures and failures over 24 hours. From Figure 3.17(a) it's obvious that the probability of data loss increases with the percentage of simultaneous failures. At approximately 0.6% of the total drive population failed the PDL of Trinity declustered

parity and dRAID reach 100%, while O-SODP experiences 32.3% PDL and G-SODP has 23% PDL. RAID again tolerates the most simultaneous failures with a 2.6% chance of data loss. Figure 3.17(b) shows the probability of data loss with the failures distributed over 24 hours. We see that the SODP has the lowest chance of data loss due to high fault tolerance and fast rebuilds. Even in the case of 1% failures, O-SODP and G-SODP have 1.2% and 1.05% PDL, respectively. Because no rebuilds completed in 24 hours RAID experienced the same PDL as Figure 3.17(a) while both Trinity declustered parity and dRAID shows a great reduction in PDL due to their faster rebuild performance.

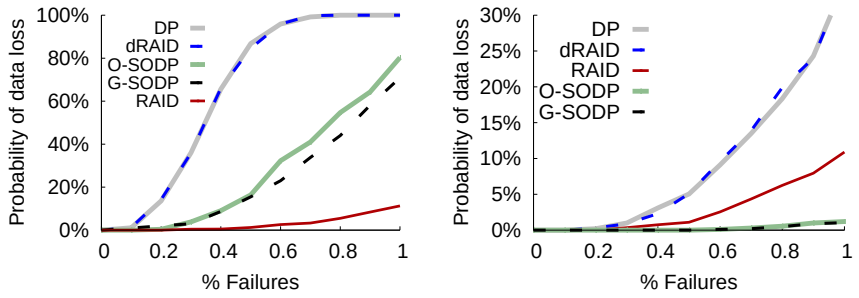

Figure 3.17: **Comparison of the probability of data loss under simultaneous failures and failure burst over 24 hours with varying failure burst sizes.** *The probability of data loss varied with the percentage of failed disks in a population of 11000 drives. During instantaneous failures in the left figure traditional declustered parity schemes experience a 100% chance of data loss once approximately 0.6% of the drives have failed. When the failures are distributed over 24 hours SODP schemes exhibit a less than 1.2% chance of data loss.*

Effect of Burst Window Finally, we investigate how the time window over which the failures occur affects the probability of data loss. Figure 3.18 shows the probability of data loss in 11,000 drives in the presence of 1% failures distributed over varying timespans (using a Poisson arrival process for failures). Because 24 hours is greater than the rebuild time for RAID the PDL remains unchanged within the varying time period. However, as declustered rebuilds complete in 2-3 hours we see the PDLs of Trinity declustered parity and dRAID are reduced rapidly. Even with slower lower rebuild performance for O-SODP, G-SODP we see extremely low probabilities of data loss.

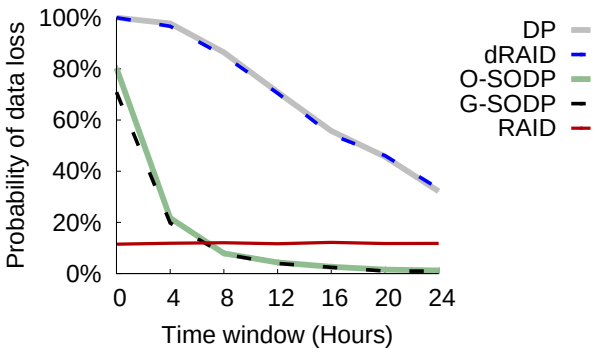

Figure 3.18: **Comparison of the probability of data loss under varying failure time window.** *The probability of data loss as 1% of drive failures are distributed over a longer time scale. As the failure window extends beyond the time to rebuild a disk we see that the SODP schemes provide better probability of data loss than non-overlapping RAID schemes. G-SODP provides slightly lower probabilities of data loss compared to O-SODP.*

Comparison of Rebuild Performance Figure 4.2.3(a) compares the disk rebuild time by injecting a random single disk failure. Trinity declustered parity and dRAID show the fastest rebuild times requiring only 2.7 and 2.8 hours. O-SODP exhibits similar rebuild performance to DP and dRAID while G-SODP requires slightly more rebuild time. Because it lacks declustering traditional raid requires greater than 24 hours to perform a rebuild. With multiple disk failures we exploit the priority reconstruction algorithm which give vulnerable stripes higher rebuild priority to avoid data loss. In our configuration (e.g., $8 + 2$), the maximal tolerable failures within a single stripeset is 2, therefore, Figure 4.2.3(b) compares the critical rebuild (e.g., stripes with 2 failures) and degraded rebuild (e.g., stripes with 1 failures) time during the reconstruction process. We observe that DP, dRAID and O-SODP have almost the same degraded rebuild time due to full declustering. A drawback of the SODP schemes is the longer critical rebuild times. Finally, RAID systems do not use priority reconstruction.

Comparison of Storage Efficiency Finally, we study the effects of storage efficiency by using different parity configurations. Since O-SODP cannot be calculated for arbitrary configurations we only compare RAID, DP, dRAID and G-SODP. In Figure 3.20 we show how the parity overhead effects the probability of data loss for differing parity overheads. Generally we see that additional

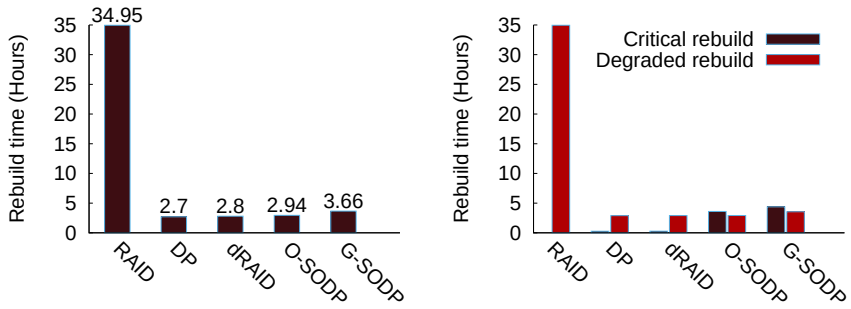

Figure 3.19: **Comparison of rebuild times.** *The left figures shows the rebuild times for single disk failure and the right figure show the rebuild times for two disk failures*

parity overhead is moderately effective during failure bursts, but that during failures distributed over time even extremely low overhead G-SODP configurations provide low probabilities of data loss.

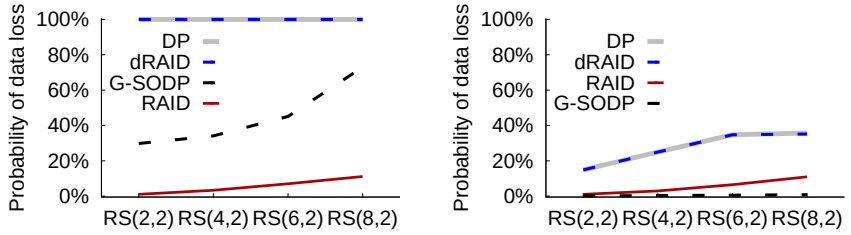

Figure 3.20: **Comparison of the probability of data loss under simultaneous failures and burst failures over 24 hours by varying storage overheads.** *The probability of data loss for RAID, DP, dRAID, G-SODP under simultaneous failures with varying parity overheads. While (a) shows that additional parity overhead is moderately effective at improving data loss probabilities during burst failures, (b) shows that G-SODP provides low probabilities of data loss while using low parity overheads.*

3.3 Conclusion

In our re-examination of declustered parity data placement schemes it is apparent that existing declustered parity data protection schemes are not designed to tolerate the correlated failure bursts becoming increasingly common in cloud and HPC data centers. To that end, we have proposed adding 2 new criteria to the design principles for declustered parity designs:

- Maximizing the number of simultaneous disk failures tolerated without increasing parity overhead, and

- Minimizing disk rebuild time by balancing parity stripes across all disks.

To balance both of these criteria equally we proposed Single-Overlap Declustered Parity, a data placement scheme that minimizes rebuild time and tolerates more failed disks than existing declustered parity designs. However, in order to build a flexible algorithm to generate SODP stripesets it became necessary to relax the single-overlap requirement and allow a small number of stripesets that do not overlap all other stripesets. Surprisingly, this more flexible algorithm demonstrated the lowest data loss during failures occurring within small windows of time. In our experiments we showed that adding disks to a declustered placement group increases the probability of a data loss event by a greater than linear amount when faced with correlated failures. But the additional disk results in a less than linear improvement in rebuild time because the amount of data being rebuilt remains fixed even as the rebuild rate is proportionally increased. Thus a data protection scheme, such as Greedy Single-Overlap Declustered Parity, which is designed to trade small amounts of rebuild performance in order to tolerate a large number of disk failures can reduce the probability of data loss substantially (30x in some of our test configurations).

CHAPTER 4

FRACTIONAL-OVERLAP DECLUSTERED PARITY: EVALUATING RELIABILITY FOR STORAGE SYSTEMS

4.1 Fractional-Overlap Declustered Parity

Traditional declustered parity originally used Balanced Incomplete Block Designs (BIBD(v, k, λ)) [36], to distribute parity stripes of size k over v disks (e.g., declustered layout), where λ is the number of disk sets that each pair-wise combination of disks appears in and $\lambda \geq 1$.

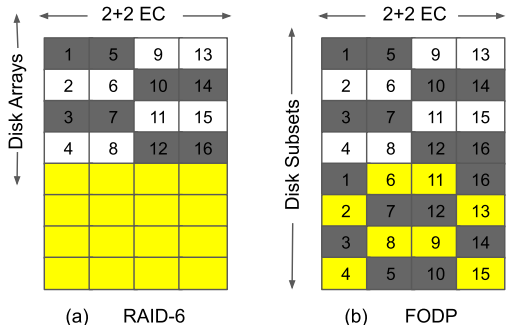

Figure 4.1: **Comparison of data layout in RAID-6 and FODP.** $2+2$ *erasure code (EC) represents 2 data and 2 parity blocks configuration, which can tolerate up to 2 failures. RAID-6 separates disks into 4 independent disk arrays while FODP comprises 8 disk subsets.*

When λ is at least 1 it ensures that all disks are able to equally participate in rebuilds because every disk has data overlapping with every other disk. For example, a declustered parity scheme that maps 2+2 stripes into a 16 disk enclosure, a λ value of 1 ensures that when a single disk fails the 15 remaining disks all participate equally in reading the surviving blocks of each degraded stripe to rebuild the failed disk. Less obvious is that the lower the λ value the more total failures that can be tolerated. Figure 4.1(a) illustrates this property using 16 disks and 4 non-overlapping 2+2 RAID6 arrays (i.e. the minimum possible λ). In this case up to 8 disks can fail without data loss assuming that those failures are distributed evenly across RAID groups. Motivated by these observations, we propose *fractional-overlap declustered parity (FODP)* $\lambda < 1$, which relaxes the overlap constraint and allows disks to overlap less than once with every other disk. As a result,

many pair-wise combinations exist, but a few pair-wise combinations are not generated. This $\lambda < 1$ scheme enables a tradeoff between rebuild performance and tolerating more disk failures by adjusting λ between the minimum and 1. Figure 4.1(b) re-organizes the RAID-6 data layout by adding 4 declustered disk subsets (e.g., the last 4 rows) in yellow and the 8 shaded failures are still tolerated, but each disk in FODP is involved in two disk subsets (i.e. 6 disks are able to participate in the reconstruction), which make the rebuild 2x faster than the RAID protection scheme.

	λ_{min}	$\lambda < 1$	$\lambda = 1$	$\lambda > 1$
Rebuild Efficiency	L	M	H	H
Fault Tolerance	H	H	M	L

Table 4.1: λ **comparison.** *H represents high, M represents medium, and L represents low.*

Table 4.1 compares the trade-off space for different λ. In the case of λ_{min}, each disk is involved in a single disk subset, which is identical to a clustered RAID protection scheme with high failures tolerance and slow disk rebuild. The case of $\lambda < 1$ represents FODP, where placement groups only partially overlap which brings moderate rebuild performance and high fault tolerance. In the case of $\lambda = 1$, it is a special type of full declustering [51] that tolerates more disk failures compared to the case of $\lambda > 1$ with identical rebuild performance. Lastly, $\lambda > 1$ is widely used in declustered parity (DP) schemes and tolerates the fewest disk failures.

4.1.1 FODP Construction

One practical limitation for $\lambda = 1$ DP placement schemes is that they are hard to generate and are impossible for many disk configurations (e.g. 8+2 parity with a 20 disk enclosure). Because FODP relaxes the traditional constraint and allows $\lambda < 1$ it is able to achieve tradeoffs similar to the ideal DP configuration while also supporting more diverse parity schemes and disk enclosure sizes. In order to describe our latin squares approach of constructing FODP layouts we first quantify the

overlap fraction λ for each disk as below.

$$\lambda = \frac{overlaps}{v - 1}$$

where each disk has (stripe-width-1) overlapping disks within a disk subset, and the *overlaps* is decided by the number of disk subsets * (stripe-width-1), which reflects the fraction that covers the remaining surviving (v-1) disks.

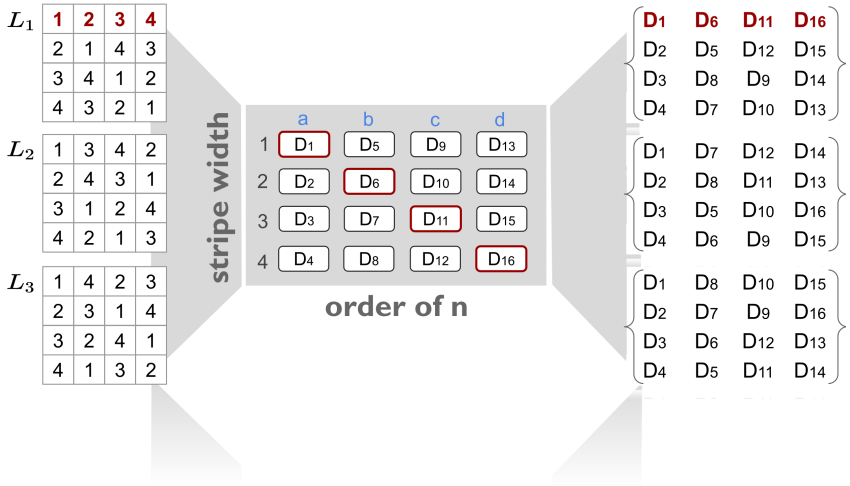

Figure 4.2: **FODP layout by mapping MOLS into the disk matrix.** *The number of rows is equal to stripe width and the number of columns is the order of n in Latin squares, where L_1, L_2, and L_3 in the left correspond to disk subsets in the right through mapping with the disk matrix.*

Definition 1. *A Latin square is a nxn array over n elements such that each element appears only once in each row and column.*

We start with a 4x4 Latin square L_1 in Figure 4.1.1, where each row and column are over elements of [1, 2, 3, 4]. To utilize the Latin square L_1, we organize the 16-disk enclosure into a disk matrix where the number of row is equal to stripe-width (e.g., rows 1, 2, 3, 4) and number of columns is equal to the Latin square order n (e.g., columns a, b, c, d). Similar to this, we specify columns a, b, c, d for Latin squares in top. By matching the 4x4 Latin square L_1 with the 4x4 disk matrix, each row is able to represent an independent 4-disk subset. For example, the first row [1, 2, 3, 4] in red represents the coordinate [(1 a), (2 b), (3 c), (4 d)], which translates into a new disk

subset [D1, D6, D11, D16]. Likewise, it's the same for the other three rows. Therefore, the Latin square L_1 corresponds to 4 non-overlapping disk subsets that cover the whole disk matrix and the *overlap fraction* λ for each disk is 3/15.

Definition 2. *Two latin squares L_1 and L_2 are called mutually orthogonal when any ordered pair of entries from each Latin square in the same row and column occurs exactly once.*

As explained above, L_1 represents 4 non-overlapping disk subsets, which would be the same for every other Latin square. To generate more disk subsets, it is necessary to use additional Latin squares. The only concern is avoiding multiple overlaps with the existing disk subsets. In other words, the L_1 related disk subsets should overlap with L_2 related disk subsets by at most one disk. Figure 4.1.1 illustrates Mutually Orthogonal Latin Squares (MOLS) L_1, L_2 and L_3, where each row overlaps with each other row by at most one overlapping element (e.g., the only overlapping element of L_{11}([1, 2, 3, 4]) and L_{21}([1, 3, 4, 2]) is element 1). The idea behind it comes from unique order pair property in Mutually orthogonal Latin squares, which guarantees the single overlap property no matter how many Latin squares we use. In essence, by applying more MOLS, we are able to linearly increase the FODP overlap fraction λ.

Theorem 1. *With any given order n, there can be at most (n-1) MOLS, with this upper bound achieved when n is a power of a prime number.*

In the order of 4, there are three Mutually Orthogonal Latin Squares (MOLS) L_1, L_2 and L_3 shown in Figure 4.1.1. With one MOLS, each disk is involved in one disk subset, which leads to overlap fraction $\lambda = 1 * 3/15$. With two MOLS, each disk is involved in two disk subsets and the corresponding overlap fraction is increased to $\lambda = 2 * 3/15$. Therefore, by applying Mutually Orthogonal Latin Squares (MOLS), we are able to linearly increase the FODP overlap fraction λ. In essence, by applying more Mutually Orthogonal Latin Squares, we are able to linearly increase the FODP overlap fraction λ.

The n-order MOLS is known to exist when n is a power of a prime number [19]. Such valid n values exist for 4, 5, 7, 8, 9, 11, 13, 16, 17, 19, 23, 25, 27, 29, 31, 32, 37, 41, 43, 47, 49, 53, 59,

61, 64, 67, 71,73, 79, 81, 83, 89, 97 in the range of 100. If the disk matrix is rectangular and the number of columns meets the valid size in MOLS then this approach ensures each disk is included in an equal number of disk subsets guaranteeing an uniform and deterministic data distribution. If the number of disks do not form a rectangular matrix we can simply remove the last row of disks and make certain to randomly substitute those disks into subsequent MOLS. In this case additional overlaps are possible and uneven data distributions may occur, but all possible configurations are supported with similar fault tolerance and rebuild characteristics to perfectly rectangular

4.1.2 FODP Model

In this section, we present a mathematical model for constructing FODP. The main idea is to maximize the stripesets with at most one overlapping disk, which ensures that each disk overlaps a single time with every other disk. To facilitate our analysis, we summarize the main notations used in this paper in Table 4.2.

Notations	Descriptions
N	the number of disk drives
f	the number of disk failures
k	data blocks per stripe
m	parity blocks per stripe
E_i	whether stripeset i exists
$S_{i,j}$	whether stripeset i contains disk j
$Y^j_{i,k}$	whether stripeset i and k overlap at disk j
S_{max}	maximal number of stripesets

Table 4.2: **Notations and variables.**

The relationship between N and $k+m$. To construct single overlap stripesets within a N-disk array, each disk overlaps a single time with every other disk, then the maximum number of stripesets per disk is

$$\frac{N-1}{k+m-1}$$

where $N-1$ is the number of remaining disks within the disk array, and $k+m-1$ is the number of remaining disks within a stripeset. Assume the first disk creates $\frac{N-1}{k+m-1}$ stripesets, then the second or other disk creates new stripesets by picking at most one disk from the existing stripesets, therefore, it poses an constraint on

$$\frac{N-1}{k+m-1} \geq k+m$$

Otherwise, it is not enough to generate any new stripesets to satisfy the single overlap property. As a result, the relationship between N and $k+m$ is as follows:

$$N \geq (k+m)(k+m-1)+1$$

The maximum number of stripesets S_{max}. To achieve the declustered layout, each disk has to be involved $\frac{N-1}{k+m-1}$ stripesets, therefore, the following relation is always true

$$S_{max} * (k+m) = N * \frac{N-1}{k+m-1}$$

This equation is to count the disk occurrence during stripesets construction in two ways. As a result, the maximum number of stripeset S_{max} can be calculated as

$$\frac{N*(N-1)}{(k+m)(k+m-1)}$$

With given S_{max} stripesets, we define a binary variable E_i to denote whether the i_{th} stripeset exists during constructing the declustered layout. Therefore, we have

$$E_i = \begin{cases} 1, & \text{if stripeset } i \text{ exists} \\ 0, & \text{otherwise} \end{cases}$$

If a stripeset exists, it can be used to place any stripe of k data blocks and m parity blocks. To maximize the fault tolerance, no two blocks in the same stripe may reside on the same disk. Therefore, it requires each stripeset to contain $k + m$ disks for holding stripes. To illustrate the relationship between stripesets and disks, we use a binary variable $S_{i,j}$ to denote whether stripeset i contains disk j as

$$S_{i,j} = \begin{cases} 1, & \text{if stripeset } i \text{ contains disk } j \\ 0, & \text{otherwise} \end{cases}$$

Recall that the total number of disks for each stripeset is $k + m$, therefore, we have the following constraint

$$\sum_{j=1}^{N} S_{i,j} = k + m, \forall i \in [1, 2, ..., S_{max}]$$

It's not always feasible to create maximum stripesets S_{max} for any given N and $k + m$. To construct the declustered layout, it does not require every stripeset to be involved, which pose another constraint on

$$\sum_{j=1}^{N} S_{i,j} = (k + m) * E_i, \forall i \in [1, 2, ..., S_{max}]$$

To generate single overlap stripesets, any two stripesets have at most one overlapping disk, which can be described as

$$\sum_{j=1}^{N} S_{i,j} * S_{k,j} = 1, \forall i, k \in [1, 2, ..., S_{max}], i \neq k$$

Note that the formulation above is a mixed-integer nonlinear programming (MINLP) problem. Fortunately, as mentioned in [91], above quadratic terms are possible to be linearized by introducing an auxiliary variable $Y_{i,k}^{j}$ as below

$$Y_{i,k}^{j} = S_{i,j} * S_{k,j}, \forall i, k \in [1, 2, ..., S_{max}], i \neq k, \forall j \in [1, 2, ..., N]$$

where $Y_{i,k}^{j}$ represents whether disk subset i and disk subset k overlaps on disk j. It can be equivalently replaced by the following linear constraint

$$0 \leq S_{i,j} + S_{k,j} - 2Y_{i,k}^{j} \leq 1, \forall i, k \in [1, 2, ..., S_{max}], i \neq k, \forall j \in [1, 2, ..., N]$$

The idea behind it is the relationship shown in Table 4.3, which uncovers the single overlap property.

$S_{i,j}$	$S_{k,j}$	$Y_{i,k}^{j}$
1	1	1
1	0	0
0	1	0
0	0	0

Table 4.3: **The relationship between $S_{i,j}$,$S_{k,j}$, and $Y_{i,k}^{j}$.**

Then, above nonlinear constraint can be rewritten in a linear form as

$$\sum_{j=1}^{N} Y_{i,k}^{j} = 1, \forall i, k \in [1, 2, ..., S_{max}], i \neq k$$

With the objective to maximize the number of disk subsets with at most one overlapping disk, the problem can be formulated as:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{S_{max}} E_i \\ \text{subject to} \quad & \sum_{j=1}^{N} S_{i,j} = (k+m) * E_i, \forall i \in [1, 2, ..., S_{max}] \\ & 0 \leq S_{i,j} + S_{k,j} - 2Y_{i,k}^{j} \leq 1, \\ & \forall i, k \in [1, 2, ..., S_{max}], i \neq k, \forall j \in [1, 2, ..., N] \\ & \sum_{j=1}^{N} Y_{i,k}^{j} \leq 1, \forall i, k \in [1, 2, ..., S_{max}], i \neq k \end{aligned} \tag{4.1}$$

4.1.3 *FODP-Plus-One*

In FODP, if more than m failures occur simultaneously within a single disk subset then all data in this subset will be lost. Therefore, increasing the overlap fraction λ, which increase the number of disk subsets, will increase the probability of data loss because the $> m$ failures will hit one of the subsets with a higher probability. Thus the total number of disk subsets is a tradeoff between the probability of data loss and the *magnitude* of data lost during a loss event. FODP is originally designed for reducing the probability of data loss with the property of $\lambda < 1$. To avoid the loss of large amounts of data during a data loss event, we further propose *FODP-Plus-One* to add a layer of parity on top of the FODP stripes.

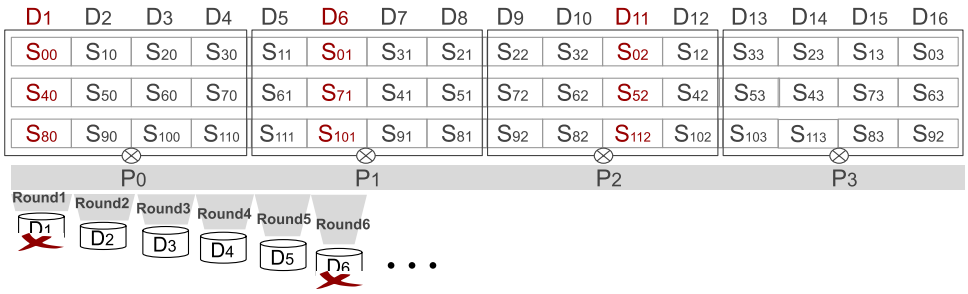

Figure 4.3: **FODP-Plus-One.** *Adding one additional parity on top of FODP and place on each disk in a round robin way.*

In Figure 4.3 we see the first blocks (e.g., $S_{00}, S_{10}...S_{11,0}$) in FODP stripes are always placed in column a (e.g., disks D_1, D_2, D_3, D_4), the second ones (e.g., $S_{01}, S_{11}...S_{11,1}$) are in column b, and so on. If we create an additional parity P_i for the i_{th} blocks in a set of FODP stripes, the parity P_i is able to further protect against FODP stripe loss by placing P_i on a disk in another column. This constructs a two-dimensional parity scheme, but uses an additional disk to store the vertical parity. Unlike existing schemes like BIBD, PDDL [77], and dRAID [45], the i_{th} blocks are always spread over the disk enclosure because the additional disk can be selected systematically. In the simplest rendition FODP-Plus-One simply places the additional parity data P_1, P_2, P_3, P_4 in round-robin fashion. A hypothetical refinement is to perform round-robin across the disks not in the original FODP disk subset, but the details of achieving balance with this approach are com-

plicated compared to the round-robin approach presented here. Although the simple round-robin placement cannot further reduce the probability of data loss, it can maximally reduce the magnitude of lost data. In this work we will limit our analysis to data protection and address performance considerations in future work.

Overall, FODP-Plus-One reduces the magnitude of data loss because the additional parity on surviving drives can recover large amounts of the lost data. Continuing our example, assume D_1, D_6, and D_{11} fail simultaneously, only the stripe S_0 including blocks S_{00}, S_{01}, S_{02}, and S_{03} on disks [D_1, D_6, D_{11}, D_{16}] will lose data. The parities P_1, P_2, P_3, P_4 on surviving disks like D_2, D_3 D_4, D_5 and so on can help recover the vulnerable stripes. As a result, the magnitude of the lost data can be significantly reduced. If we configure FODP with the maximal overlap fraction within $\lambda < 1$, the total number of disk subsets is equal to the number of disk drives v. With one additional spare drive of capacity (e.g., $v + 1$) the data lost is reduced by $(v - f)/v$ (where f is the number of disk failures).

4.2 Evaluation

In this section, we evaluate storage system reliability by addressing the following questions: (1) how do failure sizes and failure distributions impact reliability? (2) how does the overlap fraction explore the trade-offs between rebuild performance and fault tolerance? To mimic a real storage system, we consider Los Alamos National Laboratory's Campaign storage system [2] composed of 4 pods, each of which consists of 12 servers and each server is configured with two 84-disk JBODs (e.g., 168 disk drives per server). Note that to keep the FODP as similar to the Campaign storage systems 85% storage overhead we evaluate the protection schemes as if the servers have 169 disks per server and with parity configured at $11 + 2$. This is simply an artifact of trying to match the existing configuration most closely for fairer comparisons. To make these results applicable to emerging disk drive technology the simulation presented in this paper assumes that each disk drive is equipped with dual actuators and 16TB capacity, and the disk rebuild bandwidth has been set at

approximately 50M/s and the copyback bandwidth is around 400MB/s.

4.2.1 *Impact of Failures*

To study the effects of failure sizes and distributions on system reliability, we vary the percent of disk failures and set the ratio of MTBF to MTTR at 0.5 to parameterize the dense Poisson, dense Exponential, and batch cascade failures. We compare multiple parity schemes including RAID, traditional declustered parity (DP), Single-Overlap ($\lambda = 1$) Declustered Parity(SODP), FODP (e.g., with maximal overlap fraction $13/14$), and FODP+1.

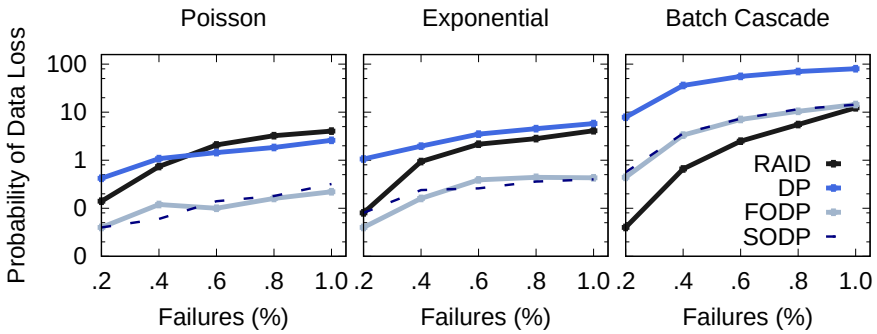

Figure 4.4: **Comparison of failure sizes and distributions.** *From left to right, the figures show the probability of data loss (PDL) resulting from Poisson and Exponential dense failures and batch cascade failures.*

Figure 4.4 shows the probability of data loss (PDL) by varying the number of failed disks from 0.2% to 1.0%. The probability of data loss increases with the percent of failures for each failure distribution and protection scheme. Interestingly, the probability of data loss in DP slightly outperforms RAID for Poisson while RAID exceeds DP in Exponential. We note that the Exponential distribution results in more outlier failures which makes RAID, with its higher fault tolerance, stand out even more. Likewise, RAID is much more efficient than DP in batch failures, where tolerating more failures is most important. Note that, FODP+1 only improves FODP with respect to the amount of lost data instead of the probability of data loss, that is why we just compare FODP and SODP. Overall, FODP experiences almost the same probability of data loss as SODP with slightly lower rebuild time and higher fault tolerance. Compared to RAID and DP, SODP

and FODP experience lower PDL in Poisson and Exponential dense failures due to higher fault tolerance and faster rebuilds while they have slightly higher PDL rates in batch cascading failures where PDL is mainly governed by fault tolerance.

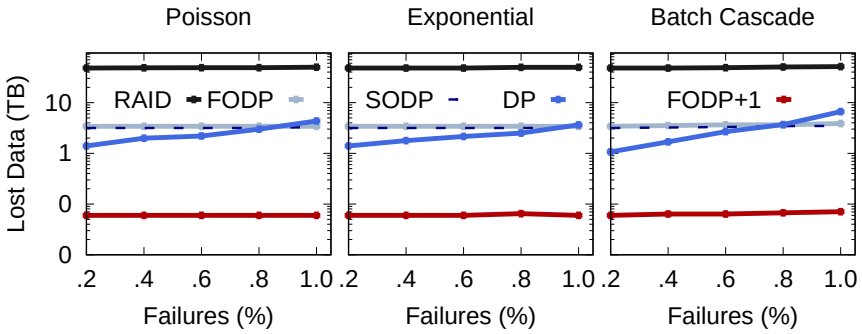

Figure 4.5: **Comparison of the amount of lost data for each incident—.** *From left to right, this figures show the amount of data loss resulting from Poisson and Exponential dense failures and batch cascade failures.*

Figure 4.5 compares the average amount of lost data when a data loss event occurs. As discussed before, reducing the number of failure domains can reduces the probability of data loss at a cost greater data lost during any data loss event. As expected, RAID experiences the highest amount of lost data, FODP and SODP have lower magnitudes of data loss with DP losing the least data. As the number of failed disks increases, the amount of lost data in RAID is slightly increased while the lost data in DP is increased and even exceeds FODP and SODP in the presence of 1% failures. Additional failures increase the probability of data loss rather than the amount of lost data, which indicates that the data loss magnitude in FODP and SODP is not highly sensitive to such small variation in failures. Unlike the other schemes FODP+1 just experiences one data loss incident each time, thus the amount of lost data remains the same in Poisson and Exponential and just slightly increases in batch failures. However, what surprise us is that FODP+1 is always the lowest (e.g.,0.06TB) in the amount of lost data. With one additional spare drive FODP+1 not only reduces the probability of data loss like FODP but also significantly reduces the magnitude of data loss.

4.2.2 *Impact of MTBF to MTTR Ratio*

To increase the system reliability, the disk MTBF should be as long as possible while the MTTR should be as short as possible. For this reason, the ratio of MTBF to MTTR directly decides the chance of data loss. To study the impact of ratios on system reliability, we vary the ratios with given MTTR to assess the probability of data loss. Figure 4.6 compares the probability of data loss by varying the ratio from 0.2 to 1.0 under 1% disk failures, which arrive at the Campaign system in a Poisson, or Exponential, or batch cascading distribution.

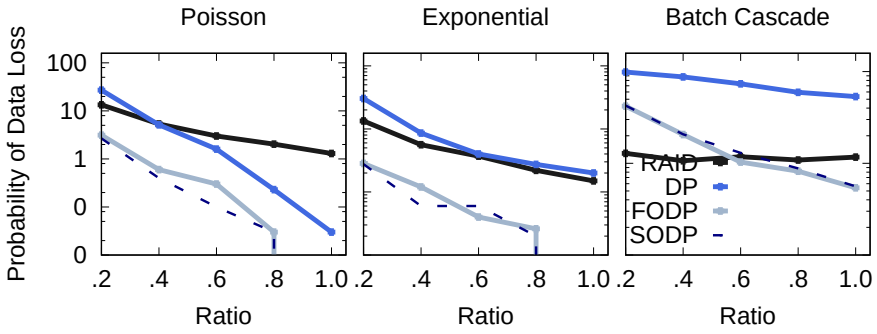

Figure 4.6: **Comparison of MTBF to MTTR ratios.** *From left to right, the figures show the probability of data loss (PDL) resulting from poisson and exponential dense failures and batch cascade failures.*

Smaller ratios mean smaller MTBFs, where being able to tolerate more failures is more important than being able to rebuild quickly. As we can see when the ratio is 0.2, RAID is always better than DP for all three failure distributions. While RAID is better than FODP and SODP in batch cascading failures, it falls behind to FODP/SODP in Poisson and Exponential distributions, where FODP and SODP are keeping decreasing and approach to zero data loss before the ratio of 1.0. On the contrary, SODP and FODP in the batch distribution experience the cascading with a faster failure rate, which makes it still falls behind in RAID under small ratios and starts to exceed RAID from the ratio of 0.6.

4.2.3 *Impact of Overlap Fraction*

To explore the effects of overlap fractions we adjust λ using FODP to compare the rebuild time and probability of data loss for Poisson, Exponential dense failures, and batch cascade failures.

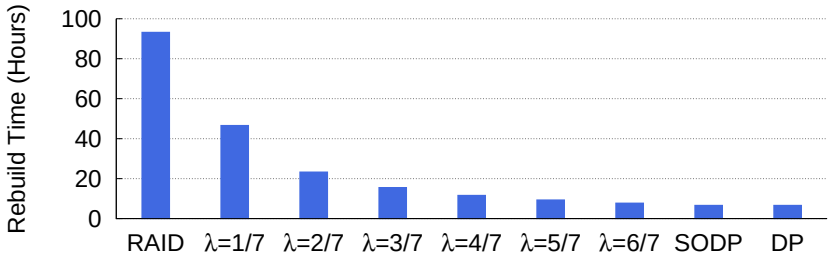

Figure 4.7: **Comparison of rebuild times.** *This figures shows the rebuild times among different data protection schemes like RAID, FODP with varying overlap fractions λ, SODP, and Declustered Parity (DP).*

Figure 4.2.3 compares the rebuild time among RAID, FODP with different overlap fractions λ (e.g., $1/7$-$6/7$), SODP and DP. As expected, there is a distinct correlation between overlap fraction and rebuild time. RAID takes the longest time (e.g., 93.2 hours) to rebuild a single disk failure. FODP with differing overlap fractions shows the diminishing rate of increase rebuild time as it approaches it's minimum rebuild time at $\lambda \geq 1$ configurations. We note that the improvement in rebuild times is not linear, but logarithmically decreasing due to the fixed amount of work being distributed over a larger number of disks.

Figure 4.8 compares the probability of data loss by varying overlap fraction λ from the minima $1/14$ (RAID) to the maxima $14/14$ (SODP) with 1% failures under various MTBF to MTTR ratios. As we can see, the probability of data loss in Poisson and Exponential failures decreases as λ increases. More interestingly we see that the PDL curves increase with λ up. to some peak, but then decrease as λ approaches 1. This phenomena is due to the time required to achieve 1% disk failures and the rebuild rate of the selected λ. With the MTBF to MTTR ratio set at 0.2 it takes 22.6 hours for 1% of the disk population to fail. At $\lambda = 5/7$ the disk rebuild time is reduced to 11 hours and the PDL begins to decrease. We see this effect in all of the Batch Cascade curves where, as the MTBF is increased, the minimum necessary rebuild performance is decreased and lower lambda values become viable though better rebuild performance still provides incremental

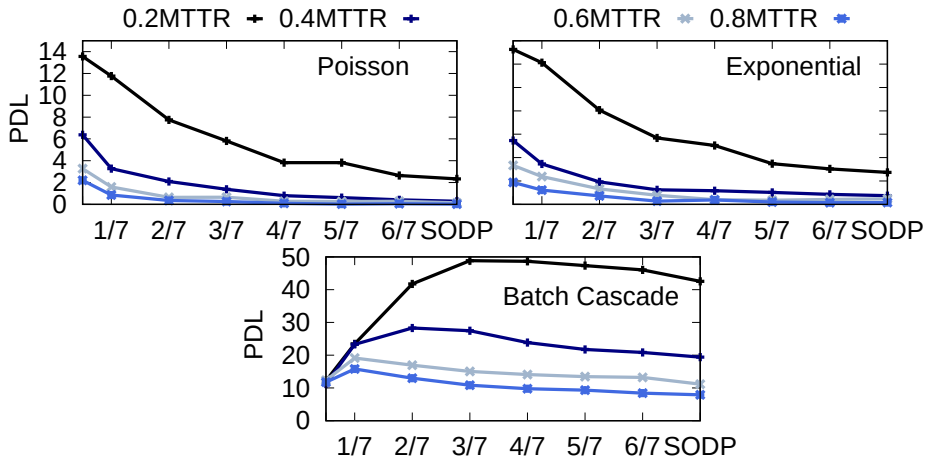

Figure 4.8: **Effects of overlap fraction on probability of data loss.** *The figures show the probability of data loss resulting from 1% disk loss due to Poisson and Exponential dense failures and batch cascade failures.*

PDL improvements.

4.3 Conclusion

Due to trends in disk drives, enclosures, and deployments we believe we are entering a period where failure events are becoming more common rather than less common. To examine that future in this paper we revisit storage system fault tolerance and rebuild performance under a diverse set of correlated failure events. To study the joint effects of these reliability characteristics this paper describes a practical and flexible declustered parity scheme, fractional-overlap declustered parity (FODP), that enables the exploration of trade-offs between rebuild performance and the number of failure domains. Our experimental results suggest that equally emphasizing the two perspectives does not achieve a midpoint in system reliability – and is sometimes the least reliable configuration. Meanwhile, FODP can reduce probability of data loss by up to 99% compared to declustered parity for various failure regimes while FODP+1 can further reduce 99% of the magnitude of data loss with a single additional spare drive capacity.

CHAPTER 5

DOING MORE WITH LESS: TIERED PARITY MYTHS AND FACTS

5.1 What Is A Better Flat Parity?

To maintain the system reliability, existing storage systems often use erasure codes (e.g., Reed Solomon (RS) Codes) to protect data with less storage overhead than replication while attaining the same fault tolerance. For example, Google ColossusFS [27] and Facebook HDFS [4] use RS(6,3) and RS(10,4) to protect data with 1.33x and 1.4x storage overhead, respectively. In general we expect storage system architects to know important parameters such as the amount of data protection that can be reasonably purchased by their organization (e.g., 3-way replication vs 20% overhead erasure coding). The common way to protect data is to use one single and uniform erasure code, or we call **flat parity (k, m)**, to balance system fault tolerance, recovery cost, and storage overhead.

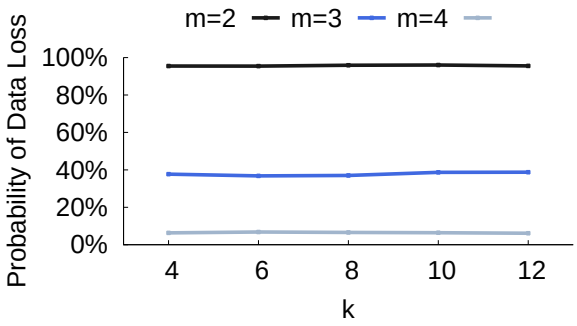

Figure 5.1: **Comparison of the probability of data loss (PDL) with different data blocks (k) and parity blocks (m).** *It shows the probability of data loss by varying the number of data blocks (k) under 0.5% simultaneous disk failures with different parity blocks (m).*

However, how to design a storage system that achieves the highest possible data reliability within the purchased storage overhead is less clear. As we can see, Figure 5.1 evaluates the system reliability in the Campaign storage file system with varying data blocks (k) and parity blocks (m), which directly decides the storage overhead. With any given m, while the smaller k brings more storage overhead, it still maintains the same probability of data loss as the larger k. On the contrary,

with any given k, the probability of data loss is inversely proportional to the value of m, which is the main factor to decide the system fault tolerance. It's obvious that larger storage overhead in our example is not necessary to bring higher system reliability. This raises an interesting question *what's the relationship between system reliability and storage overhead*. To answer this question, we start with exploring the interior trade-offs to understand how to achieve higher reliability for any given storage overhead.

5.1.1 Categorizations

The flat parity can be roughly categorized into two different data distribution schemes, one is to concentrate data and corresponding parity within small disk arrays and reserve hot-spare drives for reconstruction like the RAID protection scheme, and the other one is to distribute data, parity, and spare space uniformly over a larger disk array, typically at random [45]. Upon a failure, the RAID system is always bottlenecked by the hot-spare writes during the reconstruction, but the declustered distribution scheme enables the faster disk rebuild with more disks participating in recovery through parallel reads and writes. In essence, the declustered parity is not designed to tolerate massive failures, instead it is evolved to solve the extremely slow disk rebuild problem. To overcome the design defect, Figure 5.2 shows two common ways to improve system fault tolerance while maintaining the same storage overhead.

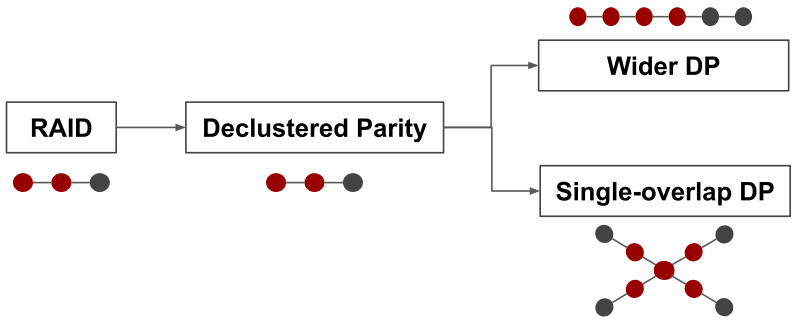

Figure 5.2: **Flat parity basics.** *With RS(2,1), to speed up the recovery process, RAID is developed into the declustered parity (DP), to further increase fault tolerance, DP can be extended to wider declustered parity (Wider DP) and single overlap declustered parity (SODP)*

Widen stripe to increase the parity (m) Unlike replication, erasure codes can increase the fault tolerance without increasing storage costs. We can observe that Reed-Solomon codes RS(2, 1) and RS(4,2) have the same storage overhead of 1.5x, but they are able to tolerate one and two failures, respectively. This increasing fault tolerance comes from increased read I/Os, where RS(2, 1) requires 2 blocks and RS(4,2) reads 4 blocks for recovering a single missing block or updating any parity block. When it comes to updating data blocks, RS(2, 1) requires updating 1 corresponding parity block but RS(4, 2) induces to update 2 parities. As a result, by widening the stripe to increase the fault tolerance, the system would suffer from longer degraded read and update complexity. Not even to say a wider stripe requires a higher computational cost. Rashmi et al [70] shows that when the number of data blocks doubled, the Reed Solomon encoding and decoding speeds are almost halved. Therefore, storage architectes have to trade-off the fault tolerance of wide stripes against the I/O amplification and complex encoding/decoding math computation.

Reduce the number of failure domains The basic reason for the declustered parity to fall short in fault tolerance lies in various stripe distributions, which allow for constructing the declustered data layout but in turn creates a large amount of failure domains. To tolerate more disk failures, one possible solution is to reduce the number of failure domains. Huan et al [51] borrow this design principle and propose single overlap declustered parity (SODP), whose idea is to have at most one overlapping disk between any two stripesets. As a result, the number of stripesets for holding stripes in the declustered parity is greatly reduced, which makes it less likely to hit one of the stripesets in the presence of $> m$ failures. Similarly, to further study the interaction between the number of failure domains and rebuild performance in declustered parity, FODP introduces *overlap fraction*, which can flexibly control the number of failure domains that in turn decides different rebuild performances. Through extensive analysis under a diverse set of correlated failure events, FODP suggests that equally emphasizing the two perspectives does not achieve a midpoint in system reliability and is sometimes the least reliable configuration.

5.1.2 *Tradeoff Spaces*

To study the tradeoff space in flat parity, Figure 5.3 compares the fault tolerance and rebuild time for single disk failure among RAID $(11+2)$, Wider DP $(16+3)$, SODP $(11+2)$, and DP $(11+2)$ within each server in the Campaign storage system. As we can see in Figure 5.3(a), RAID is able to tolerate most failures with the fewest failure domains, each of which is able to tolerate 2 failures. Although Wider DP is configured with 3 parities, the data is distributed over the whole server, which makes it only tolerate 3 failures. Likewise, SODP and DP are able to tolerate 2 failures with 2 parity blocks within each server. The only difference is that SODP constructs the fully declustered data layout with much less stripesets, which make the > 2 failures less likely to hit one of the stripesets. In other words, SODP is possible to tolerate more failures without data loss. Figure 5.3 (b) compares the rebuild time under single disk failure, where RAID takes the longest time (e.g., 93.2 hours) due to the hot spare reconstruction writing and DP takes the shortest time (e.g., 6.65 hours) by having more disks (e.g., 168 disks) participating in rebuilds parallelly. As discussed above, the Wider DP requires to read more blocks during reconstruction, which requires the rebuild time 1.5x (e.g., 9.98 hours). However, SODP improves the fault tolerance by reducing the failure domains and maintains the identical rebuild performance as DP.

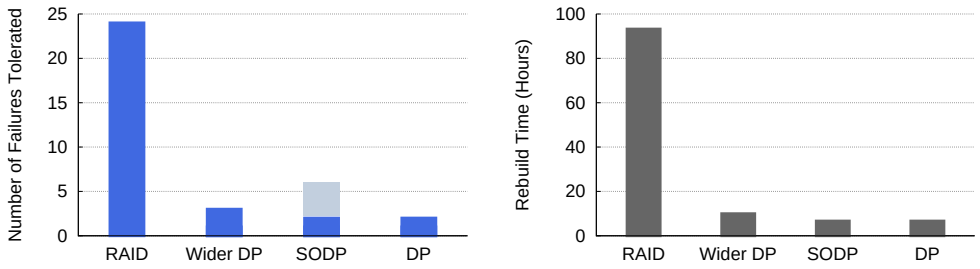

Figure 5.3: **Tradeoff Space in Flat Parity Schemes.** *fault tolerance and rebuild time comparisons within each server in Campaign storage system.*

Impact of Failures To study the fault tolerance, Figure 5.4 compares the probability of data loss by varying the number of failed disks among RAID, DP, Wider DP, and SODP in the presence of dense Poisson, dense Exponential, and batch cascade failures. We can see each failure distribution

and protection scheme increases the probability of data loss as we vary the number of failed disks from 0.2% to 1.0%. Unsurprisingly, the DP slightly outperforms RAID for Poisson while RAID exceeds DP in Exponential and batch cascade failures, where more outliers failures that cannot be recovered in time. As a result, RAID, with its higher fault tolerance, stands out even more. What interesting is that SODP is more efficient than Wider DP in batch cascade failures but slightly less efficient than Wider DP in Poisson and Exponential failure distributions. One possible explanation is that while the Wider DP slightly increases rebuild time, the given MTBF to MTTR ratio 0.5 is large enough for Wider DP (e.g., 3 fault tolerance) to recover tolerable Poisson and Exponential failures in time. When it comes to batch cascading failures, the fault tolerance in Wider DP is not enough to tolerate excessive failures (e.g. $> 0.3\%$) but SODP stands out due to the combination of higher fault tolerance and faster rebuild performance.

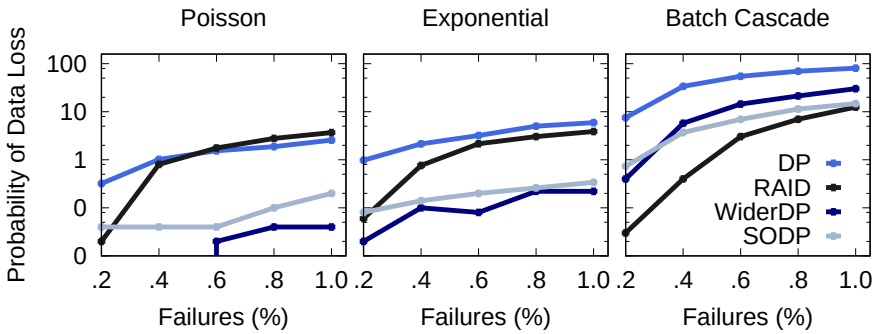

Figure 5.4: **Comparison of failure sizes and distributions.** *From left to right, the figures show the probability of data loss (PDL) resulting from dense Poisson, dense Exponential, and batch cascade failures at MTBF to MTTR ratio 0.5.*

Impact of MTBF to MTTR Ratios To study the rebuild performance, Figure 5.5 compares the probability of data loss by varying the MTBF to MTTR ratios among RAID, DP, Wider DP, and SDOP in the presence of 1% dense Poisson, dense Exponential, and batch cascade failures. It's obvious that the larger ratio, or longer MTBF, enables more disks to be recovered in time, which brings lower probability of data loss. What interesting is that the Wider DP starts to experience zero data loss event from ratio 0.4 while DP experiences zero data loss event from ratio 0.8 for Poisson failures. This difference comes from the different fault tolerances (e.g., 3 in Wider DP and 2 in

DP). Likewise, SODP experiences zero data loss event from ratio 0.6 but the overall probability of data loss is always below 1% in Poisson and Exponential failure distributions, where the Wider DP is slightly more efficient than SODP when ratio is larger than 0.2 but SODP is better than Wider DP under small MTBT to MTTR ratio 0.2. What different is that in batch cascading failures SODP is much more efficient than Wider DP for each other ratio (e.g., 0.4, 0.6, 0.8, 1.0). This implies that SODP is able to tolerate more failures than Wider DP (e.g., 16+3) when rebuild rate cannot catch up with the failure rate.

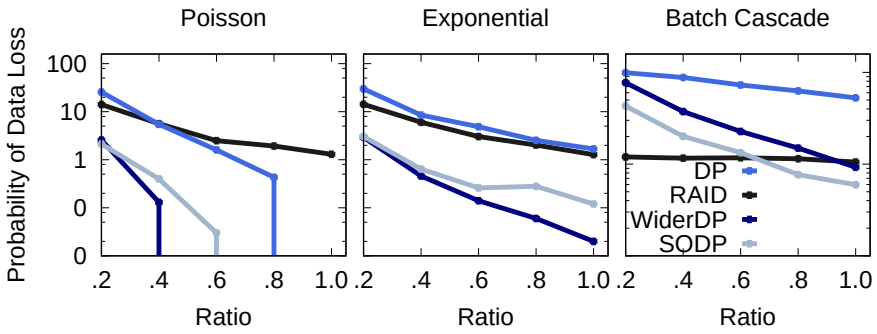

Figure 5.5: **Comparison of MTBF to MTTR ratios.** *From left to right, the figures show the probability of data loss (PDL) resulting from dense Poisson, dense Exponential, and batch cascade failures by varying MTBF to MTTR ratios from 0.2 to 1.0 under 1% disk failures.*

5.2 When Should We Add An Additional Tier?

As we discussed above, SODP outperforms Wider DP when the failure rate is higher than rebuild rate. Rather than making a tradeoff between rebuild performance and fault tolerance like Wider DP, the SODP is able to tolerate more failures while maintaining the identical rebuild performance. However, the discipline of SODP is for local disk enclosure but the array or rack failures do exist in practice. It becomes imperative to develop new methods to prevent spatial correlated failures.

5.2.1 Tiered Parity Basics

Most existing work [39, 80] advocate for rack-aware placements to distribute blocks on the same stripe across distinct racks to maximize the tolerance against rack failures. Unfortunately, this

greatly incurs large amounts of cross-rack traffics and becomes a major concern. In practice, rack failures are much rarer, which motivates some other work to trade rack-level fault tolerance for less repair traffic. For example, Patrick et.al. [94] propose the hierarchical placement to distribute the stripe blocks to fewer racks, each of which enables partial repair operations and greatly reduces the cross-rack repair traffic. While these approaches are able to tolerate rack failures, it requires the cross-rack repair traffic for any single block missing. In regard to this, Campaign storage [7] comes up multi-tier erasure codes, or **tiered parity**, which first stripes data across racks and then each rack further splits the data into blocks and computes corresponding parity blocks. In this way, the rebuild of a failed drive will take place entirely locally using the intra-rack erasure code and the system can survive the catastrophic failure of an entire rack by using the cross-rack erasure code.

5.2.2 Flat Parity or Tiered Parity

It's obvious the tiered parity is able to handle the spatial correlated failures, like the rack failures, that the flat parity cannot tolerate. However, when it comes to temporal correlated failures, it is still not clear whether the tiered parity is more efficient than the flat parity. To understand the difference, we compare the tiered parity with SODP under dense Poisson, dense Exponential and batch cascade failures as shown in Table 1.1. As mentioned in [62], failure sizes are not adequate for modeling correlated failures and may lead to suboptimal system designs. For this reason, we study the effects of both failure sizes and other important factors like MTBF to MTTR ratios that impact failure correlation.

How many disk failures we can we take? To evaluate how many disk failures we can take, Figure 5.6 varies the number of failed disks from 0.2% to 1.0%. For comparison with the same storage overhead, we configure DP and SODP with flat parity RS(11,2) and apply RS(11,1) on top of RS(12,1) in the tiered parity (Tiering). In general, the probability of data loss increases with the percentage of failures, but the effects of different failure regimes are dramatically different in

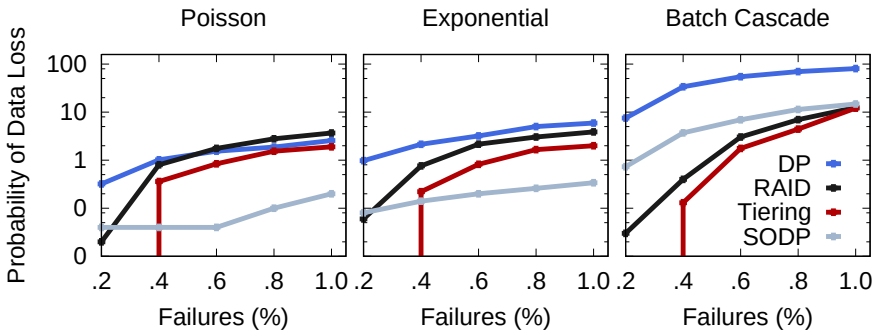

Figure 5.6: **Comparison of failure sizes and distributions.** *From left to right, the figures show the probability of data loss (PDL) resulting from dense Poisson, dense Exponential, and batch cascade failures at MTBF to MTTR ratio 0.5.*

terms of the probability of data loss. Here we simulate Poisson failures that arrive at Campaign storage system at 0.5 MTBF to MTTR ratio. Unsurprisingly, the traditional DP always experiences highest probability of data loss. RAID is less efficient than Tiering and SODP in both Poisson and Exponential failures while RAID is much more efficient than SODP but less efficient than Tiering in Batch Cascading failures, where fault tolerance is more important. This implies that Tiering provides higher fault tolerance than RAID but RAID provides higher fault tolerance than SODP. Between Tiering and SODP, what interesting is that Tiering is better than than SODP in Batch Cascade but falls behind in Poisson and Exponential. The reason behind it is that SODP has a faster rebuild performance than Tiering in Poisson and Exponential, where the rebuild performance is still a main factor to decide the probability of data loss. However under batch cascading failures, the probability of data loss is mainly governed by the fault tolerance because most failures cannot be recovered within a short cascading time window.

How do MTBF to MTTR ratios impact system reliability? To study the impact of MTBF to MTTR ratios, Figure 5.7 continue comparing the probability of data loss by varying MTBF to MTTR ratios in dense Poisson, dense Exponential failures, and batch cascade failures in the presence of 1% disk failures. As we increase the ratio from 0.2 to 1.0 in both dense Poisson and Exponential failures, the probability of data loss in SODP is always lower than that of Tiering. Overall, SODP and Tiering are more efficient than RAID and DP. However in the batch cascading

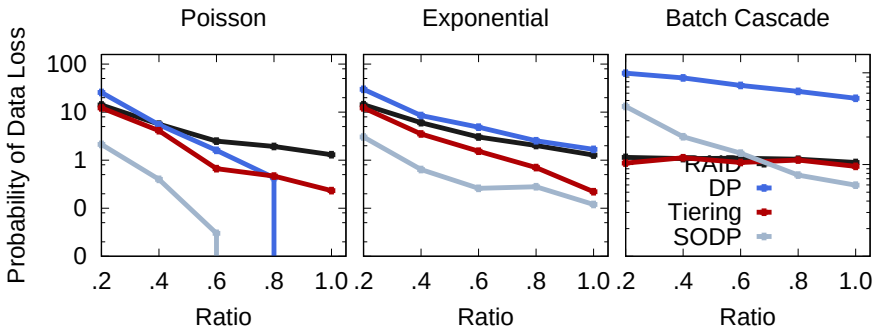

Figure 5.7: **Comparison of different MTBF to MTTR ratios.** *From left to right, the figures show the probability of data loss (PDL) resulting from dense Poisson, dense Exponential, and batch cascade failures by varying MTBF to MTTR ratios from 0.2 to 1.0*

failures, RAID is somewhere between SODP and Tiering when the ratio is less than 0.6. After that, both Tiering and SODP experience lower probability of data loss than RAID and DP. In particular, SODP is much more efficient than Tiering because the faster disk rebuild in SODP starts to work when the ratio gets larger while the slow disk rebuild time in Tiering requires a much larger MTBF to MTTR ratios.

From the above analysis, we can see if the failure rate is much faster than the rebuild rate, the ability of fault tolerance is more important. That's why Tiering can be more efficient than SODP at small MTBF to MTTR ratios in the batch cascading failures. On the contrary, if the failure rate is able to catch up with the rebuild rate, or just a little slower, the probability of data loss is a joint result of fault tolerance and rebuild performance. As a result, SODP would be better than Tiering because SODP not only tolerates more failures but also maintains the identical rebuild performance as the traditional DP. Motivated by this, we aim to design a novel tiered parity by combing Tiering and SODP to both tolerate more batch cascading failures and enable faster rebuild performance to survive dense Poisson and Exponential failures.

5.3 How Should We Design The Tiered Parity?

As we mentioned before, fault tolerance and rebuild performance are both important to system reliability. To guarantee data survival under various failure regimes, the straightforward way is to

combine SODP and Tiering to maximally explore the fault tolerance and rebuild performance. In this section, we aim at designing a novel tiered parity architecture to have one of the tiers for fast rebuild performance and one of the tiers for tolerating more failures.

5.3.1 Tiering Principles

When it comes to incorporating SODP into the tiered parity (Tiering), we have to figure out which tier we should apply SODP or both? In regard to this, we apply the alternative but flexible tool, FODP, to study the impacts of system reliability at the top and bottom tiers, respectively. This is what we call Tiered FODP (TFODP). To satisfy the configuration limitation in FODP, we configure RS(12,1) in the bottom tier and RS(5,1) in the top tier. Based on that, we evaluate the probability of data loss in the presence of 2% dense Poisson, dense Exponential, and batch cascading failures that arrive at the Campaign system at MTBF to MTTR ratio 0.5.

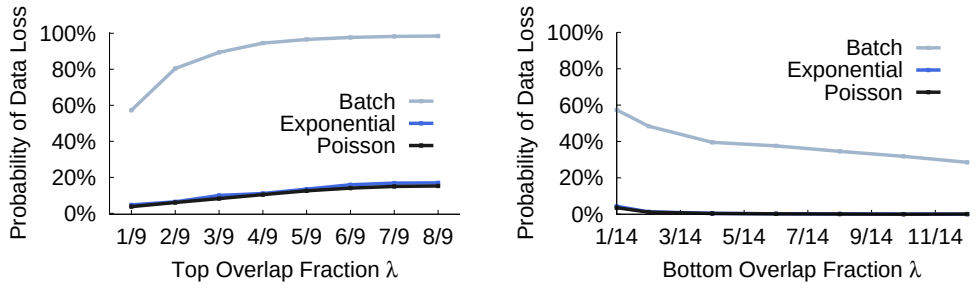

Figure 5.8: **Tiering principles.** *Comparison of the probability of data loss by varying the top and bottom overlap fractions.*

Figure 5.8 compares how the top and bottom tier overlap fractions impact the system reliability. As we can see, as the overlap fraction λ on top increases, the probability of data loss keeps increasing. On the contrary, the probability of data decreases as the overlap fraction λ in the bottom tier increases. Overall, with the same MTBF to MTTR ratios, the probability of data loss in Exponential failures is just slightly higher than that in Poisson failures, that's why the corresponding two lines in Figure 5.8 almost overlap with each other. Due to the batch cascading characteristics, the probability of data loss in batch failures are much higher than other two failure distributions. However, no matter which type of failures arrive at the system, the increasing pattern on top and

decreasing pattern in bottom help us to identify two principles that emphasize both fault tolerance and rebuild performance as below.

- The bottom tier favors larger overlap fractions for higher rebuild performance.
- The top tier uses smaller overlap fractions for higher fault tolerance.

5.3.2 Bottom Rebuild Performance

In this section, we have to answer whether the bottom tier buys us a lot in rebuild performance? Our experiments above reveal that the bottom tier is more effective with larger overlap fraction, which implies that the high rebuild performance in bottom is more important. With larger overlap in bottom, data can be distributed within each rack rather than each RAID array, which can improve rebuild performance in two dimensions: (1) it improves the single disk rebuild time by having more intra-rack disks participating in reconstruction, and (2) it alleviates the inter-rack repair traffics by having less data in each stripeset.

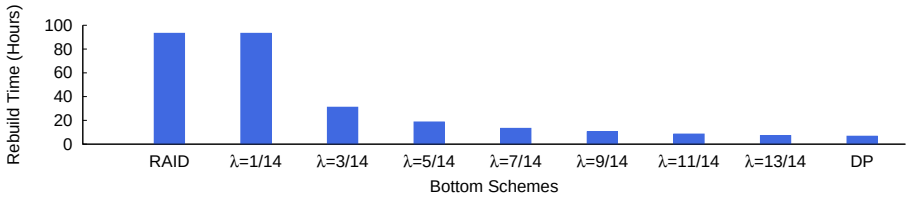

Figure 5.9: **Comparison of rebuild time for single disk failure.** *It compares the rebuild time between RAID, FODP with different overlap fraction λ, and DP in the bottom tier*

Rebuild Time To study the rebuild time for single disk failure, Figure 5.9 compares the rebuild time among RAID, FODP with varying overlap fractions λ, and DP. It's obvious that RAID takes the longest time (e.g., 93.2 hours) to rebuild a single disk failure, which is the same as the minimal overlap fraction (e.g., λ=1/14) in FODP. Both of them have each disk in an unique stripe sets exclusively. With the increase of overlap fraction λ, the corresponding disk rebuild time shows the diminishing rate due to the fixed amount of work being distributed over a larger number of disks.

As the overlap fraction approaches $\lambda > 1$, like DP, the rebuild time arrives at the minimum (e.g., 6.65 hours).

Rebuild Traffic Previously, each disk is exclusively included in one disk array, where once there is a data loss event, it means the whole disk array has to depend on the top tier to recover. As a result, the cross-rack repair traffic would be huge. By applying lager overlap fraction in bottom, each disk can be involved in multiple stripesets. When data loss events occur, only a portion of stripesets experience data loss and require the network rebuilds while other stripesets are able to rebuild locally. This is what we call **fractional rebuilds**, which significantly alleviate the cross-rack repair traffics in the presence of local data loss events. To evaluate the efficiency of the fractional rebuilds in the bottom tier, Figure 5.10 compares the average percent of local and network repair traffics for each data loss incident among RAID, FODP, SODP, and DP.

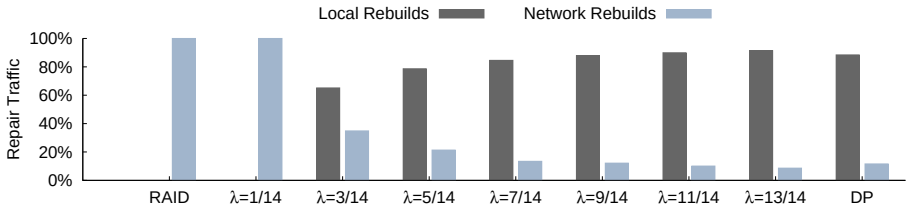

Figure 5.10: **Comparison of local and network rebuild traffics.** *This figures show the percent of traffic for each disk in the presence of 2% dense Poisson failures among RAID, FODP and DP.*

As we can see, when the overlap fraction in bottom increases, the network traffics for rebuilds gradually decreases, which leaves most rebuilds happen locally. RAID suffers from 100% network traffic once the local data loss events occur. FODP starts to experience less network traffics when the overlap fraction increases, but what interesting is that DP has more network rebuild traffics than FODP to some extent.

5.3.3 Top Fault Tolerance

Unlike the bottom tier, the top tier favors the smaller overlap fraction. This motivates us aspire to the minimal overlap fraction like RAID protection scheme. To further explore whether the top tier

buys us a lot in fault tolerance, we set up one parity overhead in top and compare different redundancy schemes, RS(47,1), RS(23,1), RS(11,1), RS(5,1), that represent different storage overheads.

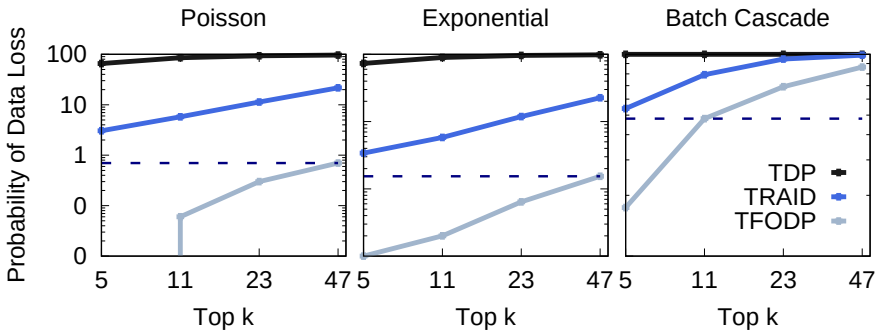

Figure 5.11: **Comparison of different top-tier fault tolerance.** *This figure shows the probability of data loss under different storage overheads (e.g., different k) for TDP, TRAID, and TFODP under 2% dense Poisson, dense Exponential, and batch cascading failures.*

Figure 5.11 compares the probability of data loss among the Tiered DP (TDP), Tiered RAID (TRAID), and Tiered FODP (TFODP), where the top tier is minimal overlap fraction (e.g., RAID) with varying k while the bottom tier applies DP, RAID, and FODP, respectively. With the same storage overhead (e.g., top same k), it is obvious the TDP always experiences higher probability of data loss than TRAID, which further has a higher chance to trigger data loss than TFODP. Overall, while the probability of data loss in TFODP is significantly increased in batch cascade failures, it is just slightly increased in dense Poisson and Exponential failures. From the perspective of probability of data loss, we notice that, in the dense Poisson and Exponential failures with RS(47,1) in TFODP, the corresponding probability of data loss is lower than that of RS(5,1) in TRAID. Likewise, when it comes to the batch cascade failures, the RS(11,1) in TFODP is even better than the RS(5,1) in TRAID. This finding tells us that TFODP is able to achieve the same fault tolerance with much lower storage overhead.The reason behind it is that bottom FODP not only speeds up the local disk rebuild performance but also tolerates more disk failures, which reduces the burden of top-tier fault tolerance. As a result, rather than tolerating more rack failures with more storage overhead on top, our Tiered FODP (TFODP) can tolerate more failures with less storage overhead.

5.4 Conclusion

This chapter revisits existing flat parity schemes that are efficient in temporal correlated failures rather then spatial correlated failures. To tolerate various catastrophic failures, we propose Tiered FODP (TFODP), which applies FODP into the tiered parity and identify a set of new design principles in the tiered parity.

- The bottom tier favors larger overlap fractions for higher rebuild performance.
- The top tier uses smaller overlap fractions for higher fault tolerance.

By applying the above two principles, we analyzed the bottom tier rebuild improvements in terms of the single disk rebuild time and rebuild traffic. And then we compared the system reliability with the same fault tolerance but different storage overheads in the top tier. Our evaluation shows that TFODP is able to achieve higher reliability with less storage overhead.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

"The impact of failure correlation on system unavailability is dramatic." — from Microsoft Research [48]

This dissertation takes critical steps towards guaranteeing data survival under correlated failures.

Step 1 is to systematically revisit previously data protection approaches, although plausible for small-scale systems, but less effective under real-world correlated failures in large-scale storage systems, resulting in system designs that suffer from unavoidable data loss events. To address this problem, this dissertation proposed single-overlap declustered parity (SODP) to tolerate more failures while maintaining the identical rebuild performance. In particular, it provide a set of new criteria (1) maximizing the number of simultaneous disk failures tolerated without increasing parity overhead, and (2) minimizing disk rebuild time by balancing parity stripes across disks, that systems should use to tolerate correlated failures.

Step 2 is to explore the tradeoff space between the reliability factors, fault tolerance and rebuild performance. This dissertation proposed fractional-overlap declustered parity (FODP) to provide a more practical and flexible declustered parity scheme with similar failure and rebuild characteristics. Unlike HACFS [87] to use two different erasure codes, our FODP can adapt to different reliability requirements with only one uniform erasure code. Furthermore, by adding one additional parity on top of FODP, our FODP-Plus-One can dramatically reduce the magnitude of lost data. Our evaluation sheds light on how to intelligently balance the fault tolerance and rebuild performance across various failure regimes.

Step 3 is to integrate SODP/FODP into the tiered parity, which layers one data protection scheme on top of another one, to protect against the catastrophic failures like rack failures and

power and cooling failures that existing flat parity schemes would definitely lose data. This dissertation identifies two design principles (1) higher rebuild performance in bottom, and (2) higher fault tolerance in top, and then proposes a novel tiered design, Tiered FODP (TODP), to tolerate more failures with less storage overhead.

6.2 Future Work

m-overlap Declustered Parity The success of single-overlap declustered parity (SODP) is to construct the declustered data layout with minimum number of stripesets, which reduces the chance of hitting one of the stripesets in the presence of correlated failures. Although it achieves the identical rebuild performance as DP in the presence of single disk failure, the rebuild performance for double disk failures is still challenging. Due to the single overlap property, any two disks are just involved in one stripeset. Therefore, the critical reconstruction (e.g., 2 failures) just engages one stripeset in disk rebuilding, which is far slower than that in the traditional decluster parity. To achieve the faster critical reconstruction, we plan to explore the feasibility of m-overlap decluster parity in future. This requires us to make an extension to the way of constructing disk matrix, where each element is supposed to contain more disks. For example, if each element in disk matrix contains two disks, then the two disk failures (e.g., critical reconstruction) are able to have all remaining surviving disks to participate in the critical reconstruction. By applying the same principle, we plan to further explore the following features: (1) various stripesets to support various stripe widths like RAIDZ [3]; (2) various overlaps to support heterogeneous disk characteristics.

Distributed Sparing With the introduction of the declustered parity, spare drives are no longer physical drives but virtual spaces composed of blocks randomly scattered across all disks in the pool. Both SODP and FODP assume distributed spare spaces are always able to write if the spare capacity is enough and corresponding disks are available. It's still not clear which disk's spare space is used to recover which stripe's block missing. FODP is able to fractionally overlap with

the remaining disks, which leaves some disks non-overlapped. In regard to this, we consider to use the non-overlapping disks and build some non-overlapping relationship with FODP stripes to specify which disk's spare space can be used to repair which's stripe's block missing. However, SODP requires each disk to overlap a single time with every other disk, which makes the distributed spare space no room to implement. This requires us a different way to consider distributed spare space in SODP.

Performance Evaluation One benefit of SODP is to achieve the identical rebuild performance as the declustered parity. To further explore that, we would like to implement SODP/FODP in ZFS file system and compare with the state-of-the-art dRAID in terms of the rebuild performance. Meanwhile, the way to distribute data and parity can affect the read/write performance. We would like to conduct a detailed evaluation of the read and write performance of SODP/FODP data protection schemes. With the given FODP/SODP stripesets, we also would like to implement some data mapping optimizations to further improve the read/write performance.

Optimal Overlap Fraction As analyzed in TFODP, the probability of data loss almost approaches zero when the overlap fraction in bottom is equal to or larger than $3/14$. This raise a question whether we need a higher overlap fraction and what the optimal overlap fraction should be. To address this problem, we plan to consider a detail analysis of local and network rebuilds. Unlike the existing tiered parity, the benefit of TFODP is the fractional rebuilds, which indicate the fraction of data needed to be recovered. With known data fractions for both rebuilds, the optimal overlap fraction could be affected by the ratio of local bandwidth and network bandwidth. The goal is to have both rebuilds to finish simultaneously. However, the practical available local and network bandwidths are decided by many factors like application workloads. It's more challenging to decide the optimal overlap fraction and I would like to conduct comprehensive evaluations to study the relationship between the optimal overlap fraction and the practical bandwidths.

Failure-aware Stripesets Our current work focus on reducing the number of failure domains (e.g., stripesets) to protect against the correlated failures without knowing any underlying failure characteristics. We consider to construct failure-aware stripesets to avoid the possible failures to occur within the same stripeset. Specifically, we mainly consider the followings failures scenarios:

- Large storage systems will likely include multiple batches of manufactured components at initial deployment, and that heterogeneity will increase over the life of the system as components are replaced. When heterogeneous disk characteristics [34, 50, 37] coexist on the same storage system, we must explicitly select which disks could comprise of a stripeset to avoid the batch cascading failures.

- [59] emphasizes that the location information because the disks in close spatial neighborhood will be affected by the same environment factors such as humidity and temperature, and experience similar vibration level, which is known to affect the disks reliability. Similarly, CoFaCTOR [54] observes that disks in the last row are more likely to encounter failures than those in the front rows.

Bottom-aware Tiered Parity The design and implementation of current tiered parity [46, 5] is to exploit independent top and bottom encodings, where both of them are unaware of each other. The way in which the redundant information is computed in top can be categorized into coarse-grained encoding. And then the coarse-grained encoded large data blocks are distributed across servers, each of which continues to divide into small blocks and encode in bottom. However, this design requires the whole large block in top to be recovered when the bottom data loss is caused by a small number of small bottom blocks. This coarse-grained encoding in top results in more unnecessary repair traffic. To solve this problem, we can use fine-grained encoding [56] in top, which is to take the bottom redundancy into consideration. In particular, the data interleaving in top can be divided into different groups by awaring of the bottom redundant information. Each group still exploits the same way to encode but the size of data units is much smaller. This brings

the advantages (1) the speed of encoding in top could be faster because of the small data units; (2) the bottom receives the small units from the top without data dividing process during encoding; (3) the repair traffic in top can be minimized by enabling small units recovery rather than a whole large block.

REFERENCES

[1] Aws service level agreements (slas). https://aws.amazon.com/legal/service-level-agreements/.

[2] Campaign storage system. https://storageconference.us/2017/Papers/CampaignStorage.pdf.

[3] Creating a raid-z storage pool. https://docs.oracle.com/cd/E53394_01/html/E54801/gcvjg.html.

[4] Facebooks erasure coded hadoop distributed filesystem (hdfs-raid). https://github.com/facebookarchive/hadoop-20.

[5] Hierarchical erasure coding: Making erasure coding usable. https://www.snia.org/sites/default/files/SNIA_Hierarchical_Erasure_Coding_Final.pdf.

[6] Redundant arrays of inexpensive disks (raids). http://pages.cs.wisc.edu/~remzi/OSTEP/file-raid.pdf.

[7] Tiered parity: When flat doesn't fit. https://permalink.lanl.gov/object/tr?what=info:lanl-repo/lareport/LA-UR-17-23986.

[8] Trinity storage system. https://www.lanl.gov/projects/trinity/about.php.

[9] The z file system (zfs). https://www.freebsd.org/doc/handbook/zfs.html.

[10] R Julian R Abel, Andres E Brouwer, Charles J Colbourn, and Jeffrey H Dinitz. Mutually orthogonal latin squares (mols). *The CRC handbook of combinatorial designs*, 2:160–193, 1996.

[11] Guillermo A Alvarez, Walter A Burkhard, and Flaviu Cristian. Tolerating multiple failures in raid architectures with optimal storage and uniform declustering. In *ACM SIGARCH Computer Architecture News*, volume 25, pages 62–72. ACM, 1997.

[12] GA Alverez, Walter A Burkhard, LL Stockmeyer, and Flaviu Cristian. Declustered disk array architectures with optimal and near-optimal parallelism. In *Proceedings. 25th Annual International Symposium on Computer Architecture (Cat. No. 98CB36235)*, pages 109–120. IEEE, 1998.

[13] Guillaume Aupy, Yves Robert, and Frédéric Vivien. Assuming failure independence: are we right to be wrong? In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 709–716. IEEE, 2017.

[14] Mary Baker, Mehul Shah, David SH Rosenthal, Mema Roussopoulos, Petros Maniatis, Thomas J Giuli, and Prashanth Bungale. A fresh look at the reliability of long-term digital storage. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, pages 221–234, 2006.

[15] Leonardo Bautista-Gomez, Ana Gainaru, Swann Perarnau, Devesh Tiwari, Saurabh Gupta, Christian Engelmann, Franck Cappello, and Marc Snir. Reducing waste in extreme scale systems through introspective analysis. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 212–221. IEEE, 2016.

[16] Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel. Finding a needle in haystack: Facebook's photo storage. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, pages 47–60, Berkeley, CA, USA, 2010. USENIX Association.

[17] Elwyn Berlekamp. *Algebraic coding theory*. World Scientific, 1968.

[18] Mario Blaum, Jim Brady, Jehoshua Bruck, and Jai Menon. Evenodd: An efficient scheme for tolerating double disk failures in raid architectures. *IEEE Transactions on computers*, 44(2):192–202, 1995.

[19] Raj Chandra Bose and Sharadchandra S Shrikhande. On the construction of sets of mutually orthogonal latin squares and the falsity of a conjecture of euler. *Transactions of the American Mathematical Society*, 95(2):191–209, 1960.

[20] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 143–157, 2011.

[21] John Chandy and AL Narasimha Reddy. Failure evaluation of disk array organizations. In *[1993] Proceedings. The 13th International Conference on Distributed Computing Systems*, pages 319–326. IEEE, 1993.

[22] Peter M Chen, Edward K Lee, Garth A Gibson, Randy H Katz, and David A Patterson. Raid: High-performance, reliable secondary storage. *ACM Computing Surveys (CSUR)*, 26(2):145–185, 1994.

[23] Yu Lin Chen, Shuai Mu, Jinyang Li, Cheng Huang, Jin Li, Aaron Ogus, and Douglas Phillips. Giza: Erasure coding objects across global data centers. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 539–551, Santa Clara, CA, July 2017. USENIX Association.

[24] Asaf Cidon, Robert Escriva, Sachin Katti, Mendel Rosenblum, and Emin Gun Sirer. Tiered replication: A cost-effective alternative to full cluster geo-replication. In *2015 {USENIX} Annual Technical Conference ({USENIX}{ATC} 15)*, pages 31–43, 2015.

[25] Asaf Cidon, Stephen Rumble, Ryan Stutsman, Sachin Katti, John Ousterhout, and Mendel Rosenblum. Copysets: Reducing the frequency of data loss in cloud storage. In *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pages 37–48, San Jose, CA, June 2013. USENIX Association.

[26] Asaf Cidon, Ryan Stutsman, Stephen Rumble, Sachin Katti, John Ousterhout, and Mendel Rosenblum. Mincopysets: Derandomizing replication in cloud storage. In *Proc. 10th USENIX Symp. NSDI*, pages 1–5, 2013.

[27] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Dale Woodford, Yasushi Saito, Christopher Taylor, Michal Szymaniak, and Ruth Wang. Spanner: Google's globally-distributed database. In *OSDI*, 2012.

[28] Peter Corbett, Bob English, Atul Goel, Tomislav Grcanac, Steven Kleiman, James Leong, and Sunitha Sankar. Row-diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pages 1–14. San Francisco, CA, 2004.

[29] Alexandros G Dimakis, P Brighten Godfrey, Yunnan Wu, Martin J Wainwright, and Kannan Ramchandran. Network coding for distributed storage systems. *IEEE transactions on information theory*, 56(9):4539–4551, 2010.

[30] Daniel Ford, François Labelle, Florentina Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. Availability in globally distributed storage systems. 2010.

[31] John Fragalla. Improving lustre ost performance with clusterstor gridraid. In *2014 HPCAC Stanford HPC & Exascale Conference*, 2014.

[32] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[33] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.

[34] Kevin M Greenan, Ethan L Miller, and Jay J Wylie. Reliability of flat xor-based erasure codes on heterogeneous devices. In *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 147–156. IEEE, 2008.

[35] Kevin M. Greenan, James S. Plank, and Jay J. Wylie. Mean time to meaningless: Mttdl, markov models, and storage system reliability. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'10, pages 5–5, Berkeley, CA, USA, 2010. USENIX Association.

[36] Marshall Hall. *Combinatorial theory*, volume 71. John Wiley & Sons, 1998.

[37] Eric Heien, Derrick Kondo, Ana Gainaru, Dan LaPine, Bill Kramer, and Franck Cappello. Modeling and tolerating heterogeneous failures in large parallel systems. In *Proceedings of*

2011 International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–11, 2011.

[38] Mark Holland and Garth A Gibson. Parity declustering for continuous operation in redundant disk arrays. *ACM SIGPLAN Notices*, 27(9):23–35, 1992.

[39] Yuchong Hu, Patrick PC Lee, and Xiaoyang Zhang. Double regenerating codes for hierarchical data centers. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 245–249. IEEE, 2016.

[40] Yuchong Hu, Xiaolu Li, Mi Zhang, Patrick PC Lee, Xiaoyang Zhang, Pan Zhou, and Dan Feng. Optimal repair layering for erasure-coded data centers: From theory to practice. *ACM Transactions on Storage (TOS)*, 13(4):1–24, 2017.

[41] Huan Ke. Optimal Single Overlap Declustered Parity (O-SODP) Feasible Solutions. shorturl.at/hqMR0.

[42] Cheng Huang, Minghua Chen, and Jin Li. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *ACM Transactions on Storage (TOS)*, 9(1):1–28, 2013.

[43] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. Erasure coding in windows azure storage. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 15–26, Boston, MA, June 2012. USENIX Association.

[44] Cheng Huang and Lihao Xu. Star: An efficient coding scheme for correcting triple storage node failures. *IEEE Transactions on Computers*, 57(7):889–901, 2008.

[45] Isaac Huang, Eric Barton, Don Brady, Andreas Dilger, and John Carrier. draid: Declustered raid for zfs high level design, January 2016.

[46] Jeffrey Thornton Inman, William Flynn Vining, Garrett Wilson Ransom, and Gary Alan Grider. Marfs, a near-posix interface to cloud objects. 42(LA-UR-16-28720; LA-UR-16-28952), 2017.

[47] Weihang Jiang, Chongfeng Hu, Yuanyuan Zhou, and Arkady Kanevsky. Are disks the dominant contributor for storage failures? a comprehensive study of storage subsystem failure characteristics. *ACM Transactions on Storage (TOS)*, 4(3):1–25, 2008.

[48] Weihang Jiang, Chongfeng Hu, Yuanyuan Zhou, and Arkady Kanevsky. Are disks the dominant contributor for storage failures? a comprehensive study of storage subsystem failure characteristics. *ACM Transactions on Storage (TOS)*, 4(3):1–25, 2008.

[49] Saurabh Kadekodi, K. V. Rashmi, and Gregory R. Ganger. Cluster storage systems gotta have heart: improving storage efficiency by exploiting disk-reliability heterogeneity. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 345–358, Boston, MA, February 2019. USENIX Association.

[50] Saurabh Kadekodi, KV Rashmi, and Gregory R Ganger. Cluster storage systems gotta have heart: improving storage efficiency by exploiting disk-reliability heterogeneity. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 345–358, 2019.

[51] H. Ke, H. S. Gunawi, D. Bonnie, N. DeBardeleben, M. Grosskopf, T. Grov, D. Manno, E. Moore, and B. Settlemyer. Extreme protection against data loss with single-overlap declustered parity. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 343–354, 2020.

[52] Huan Ke. Surviving a disk apocalypse with single-overlap declustered parity. Santa Clara, CA, February 2020. USENIX Association.

[53] Huan Ke, Haryadi S Gunawi, David Bonnie, Nathan DeBardeleben, Michael Grosskopf, Terry Grové, Dominic Manno, Elisabeth Moore, and Brad Settlemyer. Extreme protection against data loss with single-overlap declustered parity. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 343–354. IEEE, 2020.

[54] Huan Ke, Haryadi S Gunawi, David Bonnie, Nathan DeBardeleben, Michael Grosskopf, Terry Grové, Dominic Manno, Elisabeth Moore, and Brad Settlemyer. Extreme protection against data loss with single-overlap declustered parity. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 343–354. IEEE, 2020.

[55] Osama Khan, Randal C Burns, James S Plank, and Cheng Huang. In search of i/o-optimal recovery from disk failures. In *HotStorage*, 2011.

[56] Bingzhe Li, Meng Yang, Soheil Mohajer, Weikang Qian, and David J Lilja. Tier-code: An xor-based raid-6 code with improved write and degraded-mode read performance. In *2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*, pages 1–10. IEEE, 2018.

[57] Zhipeng Li, Min Lv, Yinlong Xu, Yongkun Li, and Liangliang Xu. D3: Deterministic data distribution for efficient data reconstruction in erasure-coded distributed storage systems. In *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 545–556. IEEE, 2019.

[58] Glenn K. Lockwood, Kirill Lozinskiy, Lisa Gerhardt, Ravi Cheema, Damian Hazen, and Nicholas J. Wright. Designing an all-flash Lustre file system for the 2020 NERSC Perlmutter system. In *Proceedings of the 2019 Cray User Group (CUG)*, 2019. https://cug.org/proceedings/cug2014_proceedings/includes/files/pap118-file1.pdf.

[59] Sidi Lu, Bing Luo, Tirthak Patel, Yongtao Yao, Devesh Tiwari, and Weisong Shi. Making disk failure predictions smarter! In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 151–167, Santa Clara, CA, February 2020. USENIX Association.

[60] Ao Ma, Rachel Traylor, Fred Douglis, Mark Chamness, Guanlin Lu, Darren Sawyer, Surendar Chandra, and Windsor Hsu. Raidshield: characterizing, monitoring, and proactively protecting against disk failures. *ACM Transactions on Storage (TOS)*, 11(4):17, 2015.

[61] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, and Sanjeev Kumar. f4: Facebook's warm BLOB storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 383–398, Broomfield, CO, October 2014. USENIX Association.

[62] Suman Nath, Haifeng Yu, Phillip B Gibbons, and Srinivasan Seshan. Subtleties in tolerating correlated failures in wide-area storage systems. In *NSDI*, volume 6, pages 225–238, 2006.

[63] S. Oral, J. Simmons, J. Hill, D. Leverman, F. Wang, M. Ezell, R. Miller, D. Fuller, R. Gunasekaran, Y. Kim, S. Gupta, D. T. S. S. Vazhkudai, J. H. Rogers, D. Dillow, G. M. Shipman, and A. S. Bland. Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 217–228, Nov 2014.

[64] Jehan-François Pâris and Darrell DE Long. Using device diversity to protect data against batch-correlated disk failures. In *Proceedings of the second ACM workshop on Storage security and survivability*, pages 47–52, 2006.

[65] Jehan-François Pâris, SJ Thomas JE Schwarz, Ahmed Amer, and Darrell DE Long. Highly reliable two-dimensional raid arrays for archival storage. In *2012 IEEE 31st International Performance Computing and Communications Conference (IPCCC)*, pages 324–331. IEEE, 2012.

[66] David A Patterson, Garth Gibson, and Randy H Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 109–116, 1988.

[67] Jorge E Pezoa and Majeed M Hayat. Reliability of heterogeneous distributed computing systems in the presence of correlated failures. *IEEE Transactions on Parallel and Distributed Systems*, 25(4):1034–1043, 2013.

[68] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. Failure trends in a large disk drive population. 2007.

[69] James S Plank. Erasure codes for storage systems: A brief primer.

[70] KV Rashmi, Preetum Nakkiran, Jingyan Wang, Nihar B. Shah, and Kannan Ramchandran. Having your cake and eating it too: Jointly optimal erasure codes for i/o, storage, and network-bandwidth. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 81–94, Santa Clara, CA, February 2015. USENIX Association.

[71] KV Rashmi, Nihar B Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. A" hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 331–342, 2014.

[72] A. L. Narasimha Reddy and Prithviraj Banerjee. Gracefully degradable disk arrays. *[1991] Digest of Papers. Fault-Tolerant Computing: The Twenty-First International Symposium*, pages 401–408, 1991.

[73] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[74] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. Xoring elephants: Novel erasure codes for big data. *arXiv preprint arXiv:1301.3791*, 2013.

[75] Bianca Schroeder and Garth A Gibson. Understanding disk failure rates: What does an mttf of 1,000,000 hours mean to you? *ACM Transactions on Storage (TOS)*, 3(3):8–es, 2007.

[76] Martin Schulze, Garth Gibson, Randy H Katz, and David A Patterson. How reliable is a raid? In *COMPCON*, volume 89, pages 118–123, 1989.

[77] Thomas JE Schwarz, Jesse Steinberg, and Walter A Burkhard. Permutation development data layout (pddl). In *Proceedings Fifth International Symposium on High-Performance Computer Architecture*, pages 214–217. IEEE, 1999.

[78] Thomas JE Schwarz, Qin Xin, Ethan L Miller, Darrell DE Long, Andy Hospodor, and Spencer Ng. Disk scrubbing in large archival storage systems. In *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings.*, pages 409–418. IEEE, 2004.

[79] Zhirong Shen and Patrick PC Lee. Cross-rack-aware updates in erasure-coded data centers. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10, 2018.

[80] Zhirong Shen, Jiwu Shu, and Patrick PC Lee. Reconsidering single failure recovery in clustered file systems. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 323–334. IEEE, 2016.

[81] Galen Shipman, David Dillow, Sarp Oral, Feiyi Wang, Douglas Fuller, Jason Hill, and Zhe Zhang. Lessons learned in deploying the worlds largest scale lustre file system.

[82] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*, pages 1–10. Ieee, 2010.

[83] Jeremy Thorpe. Low-density parity-check (ldpc) codes constructed from protographs. *IPN progress report*, 42(154):42–154, 2003.

[84] Neng Wang, Yinlong Xu, Yongkun Li, and Si Wu. Oi-raid: a two-layer raid architecture towards fast recovery and high reliability. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 61–72. IEEE, 2016.

[85] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320, 2006.

[86] Ouri Wolfson, Sushil Jajodia, and Yixiu Huang. An adaptive data replication algorithm. *ACM Transactions on Database Systems (TODS)*, 22(2):255–314, 1997.

[87] Mingyuan Xia, Mohit Saxena, Mario Blaum, and David A. Pease. A tale of two erasure codes in HDFS. In *13th USENIX Conference on File and Storage Technologies (FAST 15)*, pages 213–226, Santa Clara, CA, February 2015. USENIX Association.

[88] Xin Xie, Chentao Wu, Junqing Gu, Han Qiu, Jie Li, Minyi Guo, Xubin He, Yuanyuan Dong, and Yafei Zhao. Az-code: an efficient availability zone level erasure code to provide high fault tolerance in cloud storage systems. In *2019 35th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 230–243. IEEE, 2019.

[89] Lihao Xu and Jehoshua Bruck. X-code: Mds array codes with optimal encoding. *IEEE Transactions on Information Theory*, 45(1):272–276, 1999.

[90] Nezih Yigitbasi, Matthieu Gallet, Derrick Kondo, Alexandru Iosup, and Dick Epema. Analysis and modeling of time-correlated failures in large-scale distributed systems. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 65–72. IEEE, 2010.

[91] Deze Zeng, Lin Gu, Song Guo, Zixue Cheng, and Shui Yu. Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Transactions on Computers*, 65(12):3702–3712, 2016.

[92] Ennan Zhai, Ang Chen, Ruzica Piskac, Mahesh Balakrishnan, Bingchuan Tian, Bo Song, and Haoliang Zhang. Check before you change: Preventing correlated failures in service updates. In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 575–589, 2020.

[93] Guangyan Zhang, Zican Huang, Xiaosong Ma, Songlin Yang, Zhufan Wang, and Weimin Zheng. Raid+: Deterministic and balanced data distribution for large disk enclosures. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*, pages 279–294, Oakland, CA, February 2018. USENIX Association.

[94] Mi Zhang, Shujie Han, and Patrick PC Lee. A simulation analysis of reliability in erasure-coded data centers. In *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pages 144–153. IEEE, 2017.

[95] Huijun Zhu, Peng Gu, and Jun Wang. Shifted declustering: a placement-ideal layout scheme for multi-way replication storage architecture. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 134–144, 2008.