Reliability and Power-Efficiency in Erasure-Coded Storage Systems

Technical Report UCSC-SSRC-09-08
December 2009

Kevin M. Greenan
kmgreen@cs.ucsc.edu

Storage Systems Research Center
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
`http://www.ssrc.ucsc.edu/`

UNIVERSITY OF CALIFORNIA
SANTA CRUZ

RELIABILITY AND POWER-EFFICIENCY IN ERASURE-CODED STORAGE SYSTEMS

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Kevin M. Greenan

December 2009

The Dissertation of Kevin M. Greenan
is approved:

Professor Ethan L. Miller, Chair

Professor Darrell D. E. Long

Professor Thomas J. E. Schwarz

Dr. Jay J. Wylie

Lisa C. Sloan
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

For Erica, Mom, Dad, Andie and my cat Meatball.

Acknowledgments

I would first like to thank my advisor, Ethan Miller, and the other members of my committee: Darrell Long, Jay Wylie and Thomas Schwarz. Without their insight, patience and ability to push me along, none of this work would have seen the light of day. I would also like to thank the other members of the Storage Systems Research Center, particularly Mark Storer and Andrew Leung, for collaborating and keeping me sane when times got tough.

Outside of my PhD work a great deal of people have helped me get through graduate school by providing fun and often fulfilling distractions. If it weren't for the following people (in no particular order), I would have never come this far: Herbie Lee, Philip Zigoris, Ryan Andres, Jennifer Whitney, Jon Soto, Dave Hayes, Michael Schuresko, Rishi Graham and Chris Simon. I would like to thank my mother and father for giving me support, the ability to live and opportunity to get this education. Finally, I would like to thank Erica for enduring this process. I'll make it up to you somehow.

Abstract

Reliability and Power-Efficiency
in Erasure-Coded Storage Systems

by

Kevin M. Greenan

Data reliability is paramount in modern storage systems. Such reliability is generally provided using erasure codes across storage devices. Until recently, most systems employed mirroring and single parity to tolerate device failures. Recent studies suggest that these techniques are not sufficient going forward. Recent advances in the theory of erasure codes has resulted in an abundance of codes that induce interesting tradeoffs in reliability, space-efficiency and performance. In the three parts of this thesis, we study the structural properties of erasure codes and their effects on modern storage systems. We are particularly interested in linear codes with irregular fault tolerance. While such codes offer many benefits over traditional coding techniques, reasoning about the structure of these codes is non-trivial.

In the first part of this thesis, we describe our study on the reliability of erasure codes. We have developed a generalized framework for evaluating the reliability of an arbitrary erasure code over a system configuration. In the process of studying the reliability of erasure codes, we found that many traditional modeling techniques do not extend well to multi-disk fault tolerant systems, irregular codes, latent sector faults and time dependent event rates. Our framework overcomes these obstacles and allows efficient, apples-to-apples comparison between any class of linear erasure code.

In the second part of this work, we extended the simulation framework to study the reliability of erasure-coded fragment placement in a system with heterogeneous devices. In doing so, we designed a metric that quickly orders fragment placements by reliability. The metric is used in conjunction with a brute force algorithm and a simulated annealing algorithm to efficiently find near-optimal placements. An exploratory study shows the effects of fragment placement on system reliability.

Finally, we study a property we call *reconstructability* to evaluate the potential power savings in an erasure-coded storage system. Storage contributes a non-trivial amount of energy to the ever increasing power budget of data centers. Given the various environmental and monetary consequences of power-hungry data centers, energy consumption has joined performance and reliability as a principle metric in large-scale storage systems. Here we define a novel technique in power-aware systems called *power-aware coding*, which exploits the structure of an erasure code— which is generally used to provide data reliability—to save power in a storage system. We define a minimal device activation policy for a power-aware storage system and define the properties of optimal codes under this policy. A suite of metrics are derived, which are used to compare the relative expected power savings of arbitrary linear erasure codes. The metrics and the reliability simulation framework are used to perform a rudimentary exploration of the power-space-reliability tradeoff in a system that employs power-aware coding.

Chapter 1

Introduction

The emergence of low-power archival, petabyte-scale storage systems and other novel systems motivates many interesting tradeoffs between erasure encoding schemes. While new erasure codes are being developed within both the coding theory and storage systems community, no clear winner has emerged from the performance, reliability and space efficiency tradeoff space. Obviously, both the underlying storage system and application have a huge effect on how to encode data for fault tolerance. Thus, each scheme must be thoroughly analyzed for use in a particular system. We divide erasure codes into three classes: general linear MDS codes [53], XOR-based array codes [7, 46, 68, 28, 22] and XOR-based flat codes [65, 20]. FLAT MDS codes, such as Reed-Solomon codes, exhibit optimal space-efficiency and flexible fault tolerance, but many turn out to be computationally expensive in practice. Many parity-check array codes are MDS and less computationally expensive than Reed-Solomon, but are generally 2 or 3 disk fault-tolerant. Finally, XOR-based flat codes, such as Low-Density Parity Check (LDPC) codes [30], are not generally space-optimal, but tend to be computationally inexpensive, facilitate irregular fault tolerance and interesting localization properties.

Evaluating the reliability of an erasure-coded system is quite challenging. Traditionally, Markov models are used to evaluate the reliability of such systems [66, 25, 29]. Most models assume a RAID-like setting (i.e. MDS code), independence between failures and exponentially distributed failure/rebuild. In addition, many models do not account for sector failures, do not accurately model device rebuild and assume that all devices exhibit the same failure/rebuild rates. While Markov models have provided a great deal of insight into the sensitivity of disk failure and repair on system reliability, these models generally capture an extremely simplistic view of an actual system. Instead of determining new methods of modeling the reliability of storage systems, most analysis simply extend the canonical Markov model [45]. Due to the existence of non-exponentially distributed failure rates [55, 49, 14], sector errors [56, 29, 14], heterogeneous devices and a vast difference between erasure encoding schemes, we believe that traditional reliability models, while appropriate for back-of-the-envelope comparisons, are insufficient for accurate reliability analysis.

Traditionally, performance, reliability and space-efficiency have been the primary metrics of storage systems. Recent monetary and environmental limitations have lead to efforts involving the power consumption of storage systems. Digital storage is an integral part of our society and will continue to expand. A great deal of the world's data exists on spinning media; thus, simply hosting the data requires an extraordinary amount of power. Many power saving techniques rely on device deactivation [9]. We believe that such techniques can be augmented with erasure code-aware algorithms that avoid the activation of deactivated devices. Such techniques motivate an interesting tradeoff between power efficiency, reliability and space efficiency.

1.1 Thesis Statement

The complex nature of both reliable storage systems and the erasure codes used to secure digital data make the analysis of such systems difficult and prone to error. Novel techniques are required to reason about and evaluate reliability in modern storage systems. In addition, the structural properties of the erasure codes used to provide data reliability may be used to further increase reliability and potentially avoid disk activation in a power-managed storage system.

1.2 Roadmap

There are eight distinct parts of this dissertation. Chapters 2 and 3 provide related work and background information.

Chapter 4 reviews traditional reliability modeling techniques for erasure-coded storage systems and identifies area in which these techniques fail to correctly model system reliability. A novel reliability architecture, called the High-Fidelity Reliability (HFR) Simulator, is presented in Chapter 5. Unlike traditional modeling techniques, the HFR Simulator has the ability to efficiently simulate systems under arbitrary failure/repair distributions, any linear erasure code and sector failures. The HFR Simulator is thoroughly evaluated in Chapter 6.

In Chapter 7, we develop a simple analytic model, called the Relative MTTDL Estimate (RME), that quickly estimates the reliability of a erasure-coded fragment placement across heterogeneous devices. The HFR Simulator simulator described in Chapter 5 is used to validate the analytic model.

A novel technique, called *power-aware coding*, is presented in Chapter 8. We explore how the structure of certain erasure codes affect reliability and potential power savings an erasure-coded storage system. We place bounds on the rate of codes that are likely to be useful in such a setting. Additionally, we have developed a general metrics for comparing the potential power savings of different erasure coding schemes under a variety of data reconstruction policies and disk spin-up policies. These metrics are used in conjunction with our simulation framework described in Chapter 4 to analyze the reliability-power tradeoff in power-managed, erasure-coded systems.

Chapter 2

Background and Related Work

In this chapter we review background and related work in the main areas of this dissertation: erasure-coded storage (Section 2.1), reliability (Section 2.2), placement of replicated data (Section 2.3) and reliable, power-managed systems (Section 2.4).

2.1 Erasure Coded Storage

In general, reliable storage is provided via erasure codes. Whether data is mirrored or transformed via matrix operations, almost every large-scale storage system relies on erasure codes for reliability [54, 58, 57, 33, 1]. Most systems assume the existence of an k-of-n encoding scheme to protect data without considering the underlying structure of the code. At first glance, this seems sufficient. Upon closer inspection, we find that two distinct k-of-n codes may have very different performance, fault-tolerance and reliability properties.

For clarity and consistency, we define common terms used to describe erasure-coded systems. This terminology is taken from [24]. A visual representation of these terms is presented in Figure 2.1 as an array of 4 disks that encode parity through mirroring. The unit of I/O to and from disk is commonly called a *sector*. A *code element* is the rudimentary unit of data and parity, which corresponds to a bit within a code symbol. In the simplest case, a code element corresponds to a single sector, but may contain several sectors. A *stripe* is a connected set of data and parity elements. Every data element in a stripe is necessary for computing the parity elements. Finally,

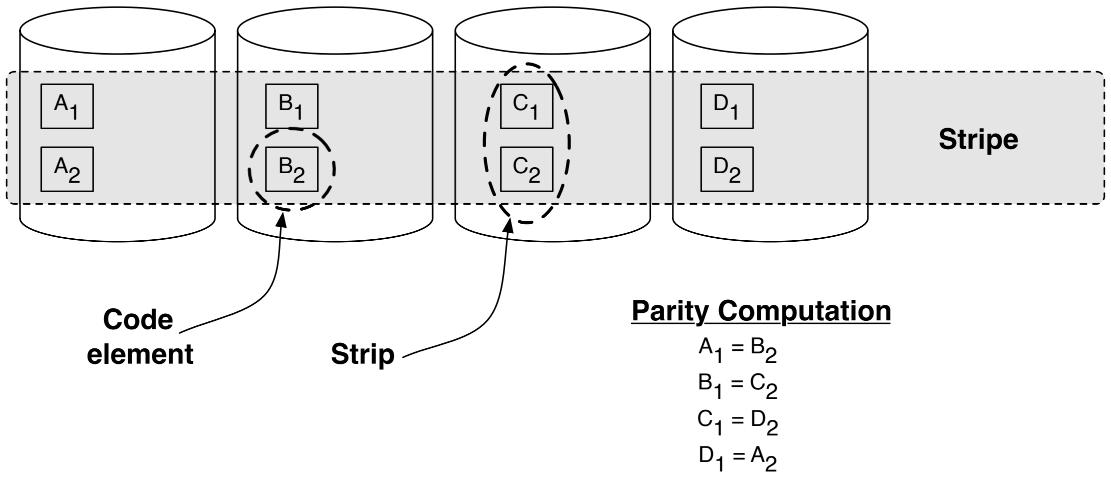


Figure 2.1: Illustration of terminology in an erasure-coded system. Here we have 4 disks that organize data and parity into stripes of height 2 (i.e. each strip contains two code elements). Parity is calculated as mirrored code elements.

a unit of storage that contains all contiguous elements from the same disk and stripe is called a *strip*. In coding theory terminology a strip is generally referred to as a *code symbol* and a stripe is a *codeword*.

We divide erasure codes into three classes : flat MDS codes, parity-check array codes and flat XOR-based codes. Flat MDS codes, such as Reed-Solomon codes [53], exhibit optimal space-efficiency and flexible fault tolerance, but turn out to be computationally expensive in practice. Most array codes are space- optimal in the number of disks and less computationally expensive than Reed-Solomon, but are typically only 2 or 3 disk fault-tolerant. Finally, flat XOR-based codes, such as Low-Density Parity Check (LDPC) codes [15], are not generally space-optimal, but tend to be computationally inexpensive and facilitate irregular fault tolerance. We only consider *systematic* erasure codes—codes that store the data and parity symbols—because their use is generally considered a necessity to ensure good read/write performance in storage systems.

A maximum distance separable (MDS) erasure code encodes k data symbols into k total symbols, $(n - k) = m$ of which are parity symbols and can tolerate all erasures of size m or less. Plank's 2005 tutorial on erasure codes is a great introduction to Reed-Solomon codes in storage systems [50]. The standard *Vandermonde Reed-Solomon* code generates each parity symbol using k Galois Fields multiplies, which are computationally more demanding than the required per parity symbol cost of $(k-1)$ XOR operations. A special Reed-Solomon code, called a *Cauchy Reed-Solomon* code, transforms each Galois field multiplication into a set of XOR operations.

An XOR-based erasure code consists of k symbols, k of which are *data symbols*, and m of which are *parity symbols* (redundant symbols). Two well known sub-classes of XOR-based erasure codes are low-density parity-check (LDPC) codes and parity-check array codes. LDPC codes trade imperfect space-efficiency for improved performance [15, 37]. Luby *et al.* [37] identified methods of constructing LDPC codes, and efficiently encoding and decoding them; such codes were originally identified by Gallager [15].

Plank and Thomason briefly surveyed LDPC code constructions for their applicability to peer-to-peer and distributed storage systems [30]. They focus on three types of constructions: systematic, unsystematic and systematic irregular repeat-accumulate codes. They focus on the *overhead factor* of these codes. That is, the fraction of total symbols required, on average, to reconstruct all of the data symbols. The codes are compared in terms of both decoding performance and overhead factor, making their study a good complement to a reliability analysis.

Parity-check array codes are specialized erasure codes for storage arrays and have a stripe height strictly greater than 1. One property the array codes have in common is an explicit mapping of disk blocks to symbols, which ensures the code can tolerate a predefined number of disk failures. Using the HoVer terminology we classify array codes as being *Ho*rizontal, *Ver*tical or both [22]. This distinction refers to the location of parity blocks and the specific blocks used to calculate the parity. A vertical code is an erasure code in which a strip contains both data and parity, while strips in a horizontal code may only contain data or parity. The simplest horizontal array code is RAID 4, which tolerates any single disk failure by calculating a single parity symbol that is the XOR of all the corresponding data symbols and storing the parity symbol on a disk designated for parity. More advanced horizontal array codes include EVENODD [7], Row-Diagonal Parity (RDP) [46] and STAR [28]. Examples of vertical codes include WEAVER [21] and X-Code [68]. Many parity-check array codes are MDS in the ratio of parity disks to total disks. For example, EVENODD is 2-disk fault tolerant and encodes parity on two disks.

GRID codes are XOR-based, highly fault-tolerant codes [35]. GRID codes are constructed from two *matched* XOR-based parity-check array codes: one representing the row-stripes (horizontal) and the other representing column-stripes (vertical). The symbols from two matched codes are mapped to the strips of an array layout. The placement of symbols is determined in one of two ways: map the horizontal code symbols first or map the vertical symbols first. After the first code is mapped to the strips, the second code is mapped to the corresponding substripes that do not contain parity. GRID codes are typically implemented using traditional array codes such as X-code, EVENODD, STAR and RAID 4. The regular structure of these codes results in relatively simple reconstruction algorithms.

We refer to non-array codes as *flat* codes: horizontal erasure codes whose codewords are comprised of exactly one row (strip height of one). Because LDPC codes are defined to be low density XOR-based codes, i.e., have parity equations with fewer than some small constant number of terms, LDPC codes are a subset of the class of flat XOR-based codes. In addition, one way to define Reed-Solomon codes is through the product of a $n \times k$ matrix and a $k \times 1$ vector. The result is an $n \times 1$ codeword vector, which is consistent with our definition of flat.

2.2 Reliability in Disk-based Storage Systems

Recent reliability analyses conclude that storage systems should have the ability to tolerate at least two disk failures [46, 54, 52, 14]. This conclusion is based on the absence of latent sector faults in traditional reliability analysis. In many cases, the event of two concurrent disk failures in the same array is quite rare. As shown explicitly in [14], a single disk failure coupled with a sector or block failure on another disk will occur with much higher frequency.

Traditionally, it was assumed that disk failure and recovery fit the memoryless, exponential distribution [45]. This assumption has propagated to a great deal of reliability studies [52, 25, 66, 56]. Recent analyses of disk failure data by Schroeder and Gibson [55], and by Pinheiro *et al.* [49] have shown that current assumptions about disk failure distributions may be incorrect. In particular, Schroeder and Gibson have shown that failure data collected from a set of 100K disks better fits a generalization of the exponential, called the Weibull distribution, which makes use of a shape parameter to model time-dependent failure rate. Given the Markov model assumption that all transitions follow an exponential distribution, using Markov models to estimate system reliability may not be sufficient going forward.

Xin *et al.* were able to model disk infant mortality in a Markov model [67]. Opposed to traditional Markov models, modeling the effects of infant mortality requires a model that accounts for time dependence: the failure rate is higher in early life than during a device's middle age. While this work made a step forward in terms of modeling time dependence in storage systems, it only applies to a specific distribution.

Bairavasundaram et al. [3] performed a failure study on latent sector faults. While specific failure rates are difficult to extract from this study, the authors show how disk age, type and size affect the frequency of latent sector faults. The authors also study the spatial locality of errors (i.e. distance on disk between errors). An alarming data corruption study performed at CERN illustrates the true impact of disk and sector failures in a real-world, high-performance storage system [44].

Elerath and Pecht recently performed a similar study that includes both latent sector and disk failures [14], which uses simulation to approximate the impact of varying failure/recovery of disks and sectors. Similar to [55], the data collected by Elerath and Pecht fit a Weibull distribution. Elerath and Pecht performed a reliability analysis on RAID 5 arrays and show that many previous reliability studies may be incorrect due to the lack of latent sector errors in the models.

Many reliability analyses consider the impact of sector redundancy within disks [10, 58]. By adding the appropriate level of sector redundancy on a single disk, we find that latent sector faults can be essentially eliminated. Disks may also be scrubbed in an effort to combat sector failures [4, 14, 56]. In this case, the rate at which disks are checked for integrity will affect the impact of latent sector faults on reliability.

A few studies build analytical models that incorporate sector failures into the reliability calculation for two or more disk fault tolerant systems [25, 10]. Unfortunately, these models do not account for partially reconstructed portions of failed disks in both rebuilding disks and determining when sector failures lead to data loss. Due to this observation, these models provide a lower bound for reliability and are only appropriate for comparing system policies.

More recently, Elerath has derived an equation for estimating the reliability of RAID 5 arrays in the face of device failures, latent defects and scrubbing [13]. The equation is based on an availability metric for the first failure (latent defect or device failure) and the hazard rate of the disk failure distribution for the second failure. The equation assumes homogeneity among device failure and repair characteristics, the Weibull distribution for failures, and only applies to singe-disk fault-tolerant systems. While the equation applies to a very specific case, it closely matches field data and simulation, providing some insight into deriving closed-form expressions for systems with non-exponentially distributed failures and repairs.

Unfortunately, there has been little work on the reliability of codes beyond single-disk fault tolerance in the systems community. Furthermore, the work that has been done is restricted to specific codes and/or assumes exponentially distributed failure and recovery rates [46, 52, 25, 16]. The difficulty in analyzing highly fault-tolerant systems lies in the rarity of data loss events. In general, data loss events are usually regarded as so-called *rare events.*, since such events only occur an extremely small percentage of the time in a storage system. Simulating rare events is straightforward for single disk fault tolerant systems [14], but becomes progressively difficult as fault tolerance increases; especially when dealing with irregular fault tolerance. While mostly unheard of in the storage systems community, the operations research community has put a great deal of effort into fast simulation models for rare events [41, 42]. A common technique is called *importance sampling.* If the simulation is viewed as a Markovian model, importance sampling adjusts the transition probabilities to increase the chances of observing a rare event. Once the rare event is observed a *likelihood ratio* based on the original and new distributions is used to transform the observation into a probability based on the original distributions. While the techniques used to implement importance sampling are straightforward, choosing the adjusted transition probabilities is difficult and must be done with care.

As a whole, current failure studies motivate the need for robust reliability mechanisms. Unfortunately, the existence of a generalized model for disk and sector failures remains open. The most probable solution will contain a unique model for each distinct class of systems. Even if we assume that analytical models are good enough for calculating reliability, unique models will most likely be necessary for each system. Given the complications that arise as the state space of a model increases, such models will be prone to human and numerical error. This observation along with the nuances present in Markov models motivates the need for a generalized framework for evaluating reliability in future storage systems.

2.3 Placement of Replicated Data

Many traditional erasure-coded placement policies focus on performance. Parity declustering is a prime example of such placement. A set of stripes of size n are placed onto $N > n$ devices with the objective of reducing repair time and its impact on performance [27, 2]. Thomasian and Blaum evaluate the reliability impact of various declustering policies in a RAID 1 organization [59]. Lian *et al.* compare the reliability of random placement vs. chained declustering in an erasure-coded brick-based architecture. They find that specific instances of random placement result in optimal reliability and propose a stripe placement scheme that achieves near-optimal reliability.

Douceur *et al.* place distinct files and their replicas onto servers in a way that maximizes the availability of the least available file [11]. They achieve this objective by using a distributed hill-climbing algorithm to swap files between servers.

Yu *et al.* study the effect of replica placement on multi-object operations and show that the availability of multi-object operations can be improved without affecting individual object availability [70].

One major drawback of all of these studies is the assumed homogeneity of the devices, placement of n replicas onto $N > n$ devices, or both. To our knowledge, no thorough reliability study of one-to-one replica to device placement in system containing heterogeneous devices exists.

2.4 Reliable Power-Managed Systems

Researchers at Colorado coined the term MAID (Massive Array of Idle Disks) [9]. The MAID system attempts to keep most of the disks inactive by employing so-called *cache disks* to store active files. When the cache disks account for a small percentage of the system, most of the disks can be placed in an inactive state, thus saving power. An extension of the MAID project uses large stripe (96 strips) Tornado codes to provide high-reliability and staggered striping to save power [63, 64]. Unfortunately, the authors chose a handful of Tornado codes based on fault tolerance and did not elaborate upon the various properties of these codes that can be exploited to save power.

Weddle *et al.* proposed PARAID, an interesting power-aware disk array architecture that switches between *gears* to account for increased or decreased load [61]. The lowest gear represents either RAID 1 or RAID 5 on a subset of the disks. Each successive gear involves activating additional disks whose blocks are replicated in the spare space of other disks. When shifting to a lower gear, the data on the inactive disks is accessed via the replicas stored on the active disks.

Hibernator relies on tiers of multi-speed disks to reduce energy, while meeting performance requirements [72]. The authors propose an algorithm that dynamically places disks into multiple tiers such that the placement minimizes energy subject to a response time constraint. During times of high utilization, more disks may be placed into high-speed mode to satisfy performance requirements. Data on a disk is organized into fixed-sized blocks, which may be migrated based on recency or popularity.

There exist a wide variety of cache-based power management solutions. The PA-LRU [71] and PB-LRU [73] algorithms attempt to increase the idle periods of inactive disks. PA-LRU evicts blocks with the minimum energy penalty (i.e. evicting it expected to result in few misses) from cache. The intuition behind the algorithm is rooted in the distribution and interval of cache misses. The PA-LRU algorithm organizes the cache into two parts : LRU0 and LRU1. Blocks with a small percentage of cold cache misses are placed into LRU1 and blocks with long access intervals are placed in LRU0. Once the cache fills up, blocks are evicted from LRU0. If LRU1 becomes empty, then blocks are evicted from LRU1 to LRU0. PA-LRU is workload dependent and must be tuned for different workloads; PB-LRU was created to avoid these problems. Given that storage workloads are not equally distributed among the disks, PB-LRU creates cache partitions for each disk. Partitions are created in a way that minimizes total energy consumption. In this case cache eviction is done on a per-disk basis, thus the cache miss sequence is easier to control.

The RIMAC [69],eRAID [60] and EERAID [34] projects are concerned with saving energy without degrading performance and use so-called transformable reads to save energy. In all cases, RAID 1 and RAID 5 are the only architectures considered. In the case of RAID 1, the disks with replicas can be spun-down and all requests are redirected to the primary disks. In an n disk RAID 5 array, a single disk can be spun down and its contents are retrieved by using the other $n-1$ disks. EERAID introduced introduced the idea of transformed reads for RAID 1 and RAID 5 which allows the system to rebuild the contents of an inactive disk from cache, other active disks, or both. EERAID also presented power-aware cache eviction algorithms that evict blocks to active devices, instead of inactive devices. RIMAC consists of a two-level collaborative cache, which exploits redundancy during I/O requests. Read requests may be transformed into an XOR operation among $n-1$ elements instead of accessing an inactive disk (or any disks if all $n-1$ elements are in cache).

Pinheiro *et al.* exploit redundancy in storage systems in an effort to reduce power consumption [48]. The authors propose a technique called *diverted access*, which segregates data and parity onto different disks. During periods of light or moderate load, the parity disks are placed into an inactive state. Parity updates are staged in non-volatile memory. The parity disks are activated during failures, high demand and to flush the contents of non-volatile ram. The authors perform a detailed analysis of their techniques and two existing data movement policies : MAID [9] and popular data concentration (PDC) [47]. Pinheiro *et al.* show that diverted access is effective when used in conjunction with existing data movement policies. Unfortunately, the cases with substantial energy savings are rather unrealistic (i.e. codes involving 14 parity disks and 1 data disk).

Harnik *et al.* study a property of low-power cloud storage systems called *full coverage* [26]. During idle periods, a subset of the disks in a large-scale storage system can be deactivated to save power. The system has reached full coverage if a subset of the devices are inactive and all of the user data is readily accessible. It is assumed that the existing data placement function for the system remains intact; thus, finding full coverage may be intractable or impossible. Auxiliary nodes are introduced to hold copies of uncovered user data as a means to reach full coverage. This work is extremely related to the balanced property of power-aware coding (cf. Chapter 8). Unlike power-aware coding, Harnik *et al.* find coverage solutions that are good enough and introduce an architectural component (auxiliary nodes) to reach full coverage. In any case, the techniques are complementary and can be used simultaneously in a cloud storage system.

Pergamum is a long-term, power-aware, reliable archival storage system [58]. The system is made up of self-contained storage bricks called *tomes*, which contain a CPU, memory, flash and a disk. Pergamum implements product codes for reliability, which enables localized repair of sector faults. In Pergamum, we showed that the use of product codes effectively renders latent sector faults non-existent. On average 5% of the disks remain active to serve writes. Each tome stores metadata and signatures in flash, which enables file search and consistency checking without activating the disk. A read request to an inactive disk results in activating the disk. In this proposal, we extend the reliability analysis, power analysis and devise techniques to improve power consumption in Pergamum.

The work we propose assumes an archival system similar to Pergamum [58]. Our energy-saving techniques are novel and are in the spirit of [48, 34, 60]. The techniques in [34, 60] only focus on RAID 1 and RAID 5, which are extremely limited in the amount of power savings achieved on read (i.e. in an n-disk array, $n-1$ symbols need to be available). The techniques in [48] only consider MDS codes and deactivation of the disks holding parity. Our approach is general in that any erasure code may be used and any device may be inactive. Our approach also allows the reconstruction of theoretically recoverable data from inactive devices. In addition, we expect our approach to further improve the cache-based approaches [71, 73, 69] and the data migration approaches [9, 47].

Chapter 3

Preliminaries

Data reliability, regardless of medium, is achieved using some form of redundancy. An *error correcting code* must be used if the presence of errors are not known a priori (i.e. sending data across a network or reading a sector off a hard drive), while *erasure codes* are typically used in situations where error locations are identified independent of the code (i.e. disk array). The difference between erasure correcting and error correcting codes is quite subtle. In fact, in many cases they are structurally the same. At a high level, error correcting and erasure codes may use the same encoding function, but have different methods of decoding.

There are four major topics that must be understood in order to fully understand the bulk of this dissertation. First, the basics of error and erasure correction are covered in Section 3.1. Galois fields and a popular MDS code code, called Reed-Solomon, are discussed in Sections 3.2 and 3.3. Finally, non-MDS , XOR-based codes and the notation used for XOR-based erasure codes is discussed in Section 3.4.

3.1 Coding for Reliability

At a high-level error correction provides the ability to both detect and correct errors, which erasure correction has the ability to correct errors. For example, assume 16 packets are sent from machine A to machine B. If the packets are encoded such that any two packet errors can be corrected, then any two packet errors can be detected and corrected. That is, if any two packets are changed between machine A and machine B, the two packets in error can be identified and fixed. Now assume the packets are encoded such that any two packet *erasures* can be corrected. Using the previous example, if two packets are changed in transmission, they cannot be identified and corrected. In this case, the packets in error must be identified through other means—usually using a signature or identification of a dropped packet— in order to perform correction.

A code is made up of so-called *codewords*, where a codeword is a vector of *symbols*. In most application, a symbol is a bit or byte-divisible block of data. A code is called *systematic* if the symbols of the codewords are separated into data and parity. For instance, RAID4 uses a systematic encoding and places each data symbol on a distinct disk and the single parity symbol on a parity disk. Many important properties of a code may be described by a few parameters : the number of data symbols per codeword, the number of parity symbols per codeword and the minimum distance between codewords. For simplicity, we will limit our discussion to linear systematic codes—the most common type of erasure/error correcting codes in storage.

We say that a (n, k, d) code, denoted $(k,n-k)$-CODE, has codewords each containing n total symbols. A codeword is made up of k data symbols and $m = n - k$ parity symbols. To encode a data block, the vector of k data symbols is mapped from a k-dimensional vector space onto an n-dimensional space such that all codewords generated from the mapping have a Hamming distance of d—meaning a codeword is at least distance d from all other codewords. The distance of two codewords is determined as the number of unequal corresponding elements. The Hamming distance between a codeword and itself is zero; all of the entries are equal to each other. The codewords $(0, 0, 1, 0, 1, 1)$ and $(1, 0, 0, 0, 1, 0)$ differ in 3 symbols, giving them a distance of 3.

The distance metric associated with a code describes the error detection and correction capabilities of the code. This is often illustrated by envisioning a sphere of radius $\frac{d}{2}$ around each codeword, which we call a *decoding sphere*. Imagine moving around the vector space by changing the symbols of a codeword. If more than $\frac{d}{2}$ distinct symbols of a codeword are changed, we move outside of the original codeword's sphere. In this case, the result either lands in another sphere or is no longer in a sphere. If exactly $\frac{d}{2}$ distinct symbols of the original codeword are changed, the result may be on the border of two spheres. In order to correctly decode an arbitrary modified codeword no more than $\lfloor \frac{d-1}{2} \rfloor$ distinct symbols can be changed. If more symbols are modified, the result may end up in another sphere, leading to an incorrect decode operation. By definition, any $d-1$ unique symbol modifications to a codeword cannot result in another codeword. Thus, a code with distance d can detect any $d-1$ symbol modifications.

Consider a codeword C from an (n, k, d) code and a word C' that is formed by changing at most $d-1$ locations in C. C can definitely be recovered from C' because every two codewords differ in at least d locations and only one codeword, C in this case, will agree on the unchanged locations. We call the known changes *erasures* and this form of correction *erasure correction*. When performing error correction the error locations are not know a priori, thus the first part of decoding determines if an error occurred and the locations of each error. Once the error locations are found, the word can be decoded.

The Singleton bound states that for any (n, k, d) code $d \leq n-k+1$, which gives an upper bound on the Hamming distance. While this bound may not provide much intuition in practice, it is used to define a very powerful set of codes. Any code that meets the Singleton bound ($d = n-k+1$) is called a *Maximum Distance Separable* code, or MDS code. In short, a MDS code is an optimally fault tolerant code for values n, k and d because it can handle any $n-k$ erasures—no code exists that can tolerate more than $n-k$ erasures.

3.2 Galois Fields

The use of Galois fields of the form $GF(2^l)$, called *binary extension fields*, is ubiquitous in a variety of area ranging from cryptography to storage system reliability. In this work, we are primarily concerned with erasure codes, which are generally encoded/decoded using Galois field operations. For completeness, we give a brief description such fields.

A Galois field $GF(2^l)$ is defined by a set of $2^l \geq 1$ unique elements that is closed under both addition and multiplication, in which every non-zero element has a multiplicative inverse and every element has an additive inverse. As with any field, addition and multiplication are associative, distributive and commutative [36]. The Galois field $GF(2^l)$ may be represented by the set of all polynomials of degree at most $l-$

1, with coefficients from the binary field $GF(2)$—the field defined over the set of elements 0 and 1. Thus, the 4-bit field element $a = 0111$ has the polynomial representation $a(x) = x^2 + x + 1$.

In contrast to finite fields defined over an integer prime, the field $GF(2^l)$ is defined over an *irreducible polynomial* of degree l with coefficients in $GF(2)$. An irreducible polynomial is analogous to a prime number in that it cannot be factored into two non-trivial factors. Addition and subtraction in $GF(2)$ is done with the bitwise XOR operator, and multiplication is the bitwise AND operator. It follows that addition and subtraction in $GF(2^l)$ are also carried out using the bitwise XOR operator; however, multiplication is more complicated. In order to multiply two elements $a, b \in GF(2^l)$, we perform polynomial multiplication of $a(x) \cdot b(x)$ and reduce the product modulo an l-degree irreducible polynomial over $GF(2)$. Division among field elements is computed in a similar fashion using polynomial division. The *order* of a non-zero field element α, $\text{ord}(\alpha)$, is the smallest positive i such that $\alpha^i = 1$. If the order of an element $\alpha \in GF(2^l)$ is $2^l - 1$, then α is *primitive*. In this case, α generates $GF(2^l)$, *i. e.*, all non-zero elements of $GF(2^l)$ are powers of α.

For small Galois fields, it is possible to calculate all possible products between the field elements and store the result in a (full) look-up table. However, this method consumes large amounts of memory—$O(n^2)$ for fields of size n. Log/antilog tables make up for storage inefficiency by requiring some computation and extra lookups in addition to the single lookup required for a multiplication table. The method requires $O(n)$ space for fields of size n and is based on the existence of a primitive element α. Every non-zero field element $\beta \in GF(2^l)$ is a power $\beta = \alpha^i$ where the *logarithm* is uniquely determined modulo $2^l - 1$. We write $i = \log(\beta)$ and $\beta = \text{antilog}(i)$. The product of two non-zero elements $a, b \in GF(2^l)$ is computed as $a \cdot b = \text{antilog}(\log(a) + \log(b)) \mod 2^l - 1$.

For a more detailed description of Galois fields and their relative performance, please refer to [18]. The general theory of finite fields is presented in [36].

3.3 Reed-Solomon Codes

Currently, Reed-Solomon is probably the most popular erasure correcting code. Reed-Solomon is used as both an error correcting and erasure correcting code; we are primarily concerned with erasure correction, thus describe erasure correction using Reed-Solomon. As we will see, Reed-Solomon meets the Singleton bound and is thus MDS .

A straightforward construction of Reed-Solomon involves Vandermode matrices. Let $\alpha_0, \alpha_1, ..., \alpha_{n-1}$ be distinct elements of the field $GF(2^l)$, where $2^l \geq n$. We define the $k \times n$ Vandermonde matrix G, where the element $g_{i,j}$ is $\alpha_i^j \forall i \in [0, n), \forall j \in [0, k)$

$$G = \begin{pmatrix} 1 & 1 & ... & 1 \\ \alpha_0 & \alpha_1 & ... & \alpha_{n-1} \\ \alpha_0^2 & \alpha_1^2 & ... & \alpha_{n-1}^2 \\ & ... & ... & \\ \alpha_0^{k-1} & \alpha_1^{k-1} & ... & \alpha_{n-1}^{k-1} \end{pmatrix}$$

The rank of G is k and any $k \times k$ submatrix of G is invertible—G^{-1} is unique. Let $\mathbf{d} = \langle d_0, d_1, ..., d_k \rangle^T$ be a vector of data symbols, which are encoded as a codeword $G\mathbf{d} = \mathbf{c}$. We obtain a $k \times k$ matrix G' by deleting $n - k$ rows of G and a vector $\mathbf{c}'$ by deleting the corresponding elements of $\mathbf{c}$ giving us $G'\mathbf{d} = \mathbf{c}' \Leftrightarrow \mathbf{d} = G'^{-1}\mathbf{c}'$. Since G^{-1} is unique, two distinct codewords can differ in at most $k - 1$ positions, otherwise the codewords are not distinct. It follows that all codewords are *at least* distance $n - (k - 1) = n - k + 1$ from one another; thus, Reed-Solomon codes meet the Singleton bound.

In storage systems, Reed-Solomon codes generally have *systematic* form. Elementary row operations are used to transform G into $(I_k|P)^T$, where I_k is the $k \times k$ identity matrix and P is called the parity submatrix. Elementary row operations will preserve the rank of the matrix and Hamming distance of the underlying code. The systematic encoding separates the data symbols and parity symbols and is computed as $(I_k|P)^T\mathbf{d} = (\mathbf{d}|\mathbf{p})^T$, where $\mathbf{d}$ are the original data elements and $\mathbf{p}$ represent the calculated parity. Since any $k \times k$ submatrix of $(I_k|P)^T$ is invertible, we can survive the loss of at most $n-k$ symbols by removing $n-k$ rows from $(I_k|P)^T$ and solving the resulting system for $\mathbf{d}$.

While Reed-Solomon codes are space-optimal for (n, k, d), encoding and decoding is rather expensive. During encoding, each parity symbol requires at least $k-1$ XOR operations and if $(n-k) > 1$, k Galois field multiplications per parity element beyond the first. Decoding begins by inverting a matrix and multiplying the k-element vector by the inverted matrix (similar to encoding). Cauchy-based Reed-Solomon codes [31, 30] trade the Galois field multiplies for additional XORs (XOR is cheaper than general Galois field multiplication methods). Cauchy matrices have properties similar to Vandermonde matrices (i.e. a sub-matrix of a Cauchy matrix is also Cauchy and every $k \times x$ submatrix of a Cauchy matrix is invertible).

3.4 Tanner Graphs and XOR-based Codes

In general, XOR-based codes do not have the rigorous construction of Reed-Solomon codes. Here, we cover systematic codes and consider systematic XOR-based code and XOR-based code to be synonymous. As the name implies, all parity computations exclusively use the XOR operator. Given a set of k data symbols, we define each of the $n-k$ parity symbols as the XOR sum of some subset of the k data symbols. We call the XOR sums for each parity symbol a *parity equation.* There exist $\left(2^k - 1\right)^{n-k}$ (not necessarily unique) systematic XOR-based codes with k data symbols and $n-k$ parity symbols.

An XOR-based code is completely specified with a Tanner graph and/or generator matrix. Both of these objects describe how to generate $n-k$ parity symbols from k data symbols. A Tanner graph is a bipartite graph, where the left vertices represent data symbols and the right vertices represent parity symbols. An edge joins a left node vertex to a right node vertex and represents membership in a parity equation. A parity symbol is computed by XORing the data from the associated data nodes adjacent to the corresponding parity node in the graph. An example Tanner graph is shown in Figure 3.1. The resulting parity equations are :

$$s_5 = s_0 \oplus s_1 \oplus s_2$$

$$s_6 = s_0 \oplus s_1 \oplus s_3$$

$$s_7 = s_0 \oplus s_2 \oplus s_3 \oplus s_4$$

If we associate disks with nodes in the graph, then the parity stored on disk 5 is the XOR sum of the corresponding data from disks 0, 1 and 2.

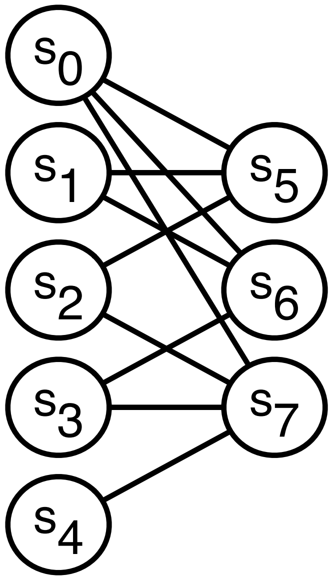


Figure 3.1: Example Tanner graph for a (5,3)-FLAT code.

Unlike Reed-Solomon codes, XOR-based codes may exhibit *irregular fault tolerance*: they tolerate certain sets of losses up to the Hamming distance of the code. For example, the example (5,3)-FLAT code in Figure 3.1 is irregular and can tolerate the loss of symbols 0 and 1, but not 4 and 7.

An equivalent definition exists for generator matrices. Let $s_0 = d_0, s_1 = d_1, s_2 = d_2, s_3 = d_3, s_4 = d_4$ be the data associated with disks 0, 1, 2, 3 and 4. The Tanner graph in Figure 3.1 can be transformed into the matrix shown in Figure 3.2. Each symbol has an associated column in the matrix. A data column is a unit vector and a parity column is a linear combination of a subset of data columns. In this case, the column for the i-th disk is a unit vector with a 1 in the i-th element. A parity column is a linear combination of the columns that correspond to the data elements involved in its parity equation. A codeword is generated by multiplying an $k \times 1$ vector of data to the $k \times n$ generator matrix.

Our definition of XOR-based codes covers two sub-classes of codes : flat XOR-based codes and array codes. A flat code associates a storage device within a unique

$$(d_0, d_1, d_2, d_3, d_4)^T \begin{array}{cccccccc} s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 \\ \end{array}$$

$$(d_0, d_1, d_2, d_3, d_4)^T \left(\begin{array}{cccccccc} s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & s_7 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \end{array}\right)$$

Figure 3.2: Example encoding of a (5,3)-FLAT using a generator matrix.

code symbol, while an array code associates multiple code symbols with a single device.

3.4.1 Minimal Erasures List

Since most systems deploy MDS codes, Hamming distance is typically used to determine fault tolerance. This makes sense because an MDS erasure code can tolerate up to a fixed number of failures. As we have shown, certain XOR-based codes exhibit irregular fault tolerance, therefore, they can tolerate failures beyond the Hamming distance of the underlying code. Wylie and Swaminathan [65] have developed the Minimal Erasures (ME) Algorithm to efficiently analyze an XOR-based code and characterize its fault tolerance. The Minimal Erasures Algorithm produces a Minimal Erasures List, which is a compact representation of a code's fault tolerance. We make extensive use of the MEL in our work.

In order to discuss the Minimal Erasures List (MEL), we must present a few definitions. A *set of erasures* is a set of erased (i.e. lost) symbols. An *erasure pattern* is a set of erasures that result in any irrecoverable data loss. Finally, an *minimal erasure* is an erasure pattern in which every erasure is necessary and sufficient for it to be an erasure pattern. The ME Algorithm generates an MEL, which is a list of all minimal erasures for a particular code. The MEL completely describes the fault-tolerance of an XOR-based code. Wylie and Swaminathan also present another object called a Minimal Erasures Vector (MEV), where the i-th element in the vector lists the number of minimal erasures of size i in the MEL. Obviously, the first non-zero entry in the MEV corresponds to the Hamming distance.

3.4.2 Terminology

Here we detail the terminology used throughout this proposal. We refer to a n symbol code with k data symbols and $(n-k)=m$ parity symbols as (k,n)-NAME, where name describes the class of code. The i-th symbol in the code is identified as s_i. A *parity bitmap* provides a compact representation of a code's parity equations, which consists of an integer for each parity equation. For example, the parity bitmap for the (5,3)-FLAT code can be derived from the generator matrix shown in Figure 3.2. Since a $k \times k$ identity matrix is used to encode the data symbols, the bitmap for each data symbol s_i is 2^i. The bitmap for a parity symbol is simply the sum of bitmaps in its parity equation. For example, the first parity symbol, s_5, has bitmap $7_{10} = 111_2 = 2^0 + 2^1 + 2^2$.

In addition, we also rely on a compact form of the Tanner graph. The Tanner graph may be described by a list of (data, parity) edges. For example, the Tanner graph for the (5,3)-FLAT code is

$$\{(s_0,s_5),(s_0,s_6),(s_0,s_7),(s_1,s_5),(s_1,s_6),(s_2,s_5),(s_2,s_7),(s_3,s_6),(s_3,s_7),(s_4,s_7)\}.$$

As an example, the MEL of the code described by the generator matrix in Figure 3.2 is $\{(s_4,s_7)$, (s_0,s_1,s_4), (s_0,s_1,s_7), (s_0,s_2,s_6), (s_0,s_3,s_5), (s_1,s_2,s_3), (s_1,s_5,s_6), (s_2,s_4,s_5), (s_2,s_5,s_7), (s_3,s_4,s_6), $(s_3,s_6,s_7)\}$, the MEV is $(0,1,10)$, the Hamming distance is 2, and all sets of erasures of size 4 lead to data loss.

Chapter 4

Estimating the Reliability of Erasure-coded Storage Systems

Erasure codes are the means by which storage systems are typically made reliable. MDS codes, such as RAID 5 and Reed-Solomon, are the most common codes used in storage systems. Recently, a wide range of special-purpose and novel erasure codes have been proposed to service the increased need for highly fault tolerant storage schemes (e.g., [30, 21, 22, 65]). Such erasure codes offer benefits traditional erasure codes cannot, such as reduced encode/decode cost, reduced small write costs, and/or localized device rebuild.

MDS codes provide optimal fault tolerance. That is, in an MDS code, k data symbols generate m parity symbols and can tolerate the loss of any m symbols. In other words, MDS codes have a Hamming distance of $d = m + 1$. Most of the proposed novel erasure codes are non-MDS and exhibit *irregular* fault-tolerance. If k data symbols generate m parity symbols, then a non-MDS code can tolerate many, but not all, erasure patterns of size d, 3, $\ldots m$, where d is the Hamming distance of the code.

Traditional reliability models were constructed with four simplifying assumptions: the only failures are whole-device failures, the use of single-disk fault-tolerant codes, devices fail at a constant rate, devices repair at a constant rate.

Until recently, single-disk fault-tolerant codes were the standard in parity-based disk arrays. At a high level, these codes were MDS codes with $m = 1$. Since only whole-disk failures are considered, keeping track of failure patterns that lead to data loss is relatively straightforward: any failure pattern involving 2 disk failures is a data loss event.

The inclusion of time dependence into reliability models increases complexity and can prohibit analytic reliability estimates. Analytic estimates can obtained through Markov models as long as failure and repair rates are constant in time. For this reason, most reliability analysis assumes constant failure and repair rates.

The main objective of this chapter is to show that traditional modeling techniques are not well-suited for accurate storage system reliability modeling. Traditional reliability analysis in storage systems has relied on Markov models constructed around single-disk fault-tolerant systems. While such models enable quick analytic reliability estimates of single-disk fault-tolerant systems, they do not extend well to multi-disk fault-tolerant systems, the inclusion of sector failures and they do not compensate for time dependence in failure and repair.

This chapter provides two major contributions. First, we review traditional Markov modeling techniques and show that they do not accommodate time dependency or easily extend to multi-disk fault-tolerant MDS codes and non-MDS codes. Second, prior models for sub-component (i.e. sector) failures are either too coarse-grained or excessively complicate reliability models. We explore the tradeoffs between the various sector failure models to show which models are best suited for a multi-disk fault-tolerant system.

We begin by reviewing standard metrics for storage system reliability: *reliability*, *unreliability* and *mean time to data loss* (MTTDL). We argue that the aggregate nature of the MTTDL makes the metric only suitable for relative comparisons, while reliability and unreliability are direct measures of system reliability. Next, we explore the canonical, 1-disk fault-tolerant Markov model for erasure-coded systems and show how these models have been extended to m-disk fault-tolerant systems. We argue that extending the canonical model to multi-disk fault-tolerant systems leads to issues in modeling rebuild, irregular codes and time dependence. Finally, we explore two dominating methods for modeling sector failures: BER and latent sector errors. Both models are typically built as extensions to the canonical Markov model. We show that each model does not fully capture the effect of sector errors and propose a more accurate model.

4.1 Measuring Reliability in Fault-Tolerant Systems

We begin our discussion by defining some of the common reliability measures in fault-tolerant systems. We assume that the system is new and operational at time $t = 0$. Let T_F be the time where the system first reaches a failure state. We assume the system has n devices, $D_0, D_1, \ldots, D_{n-1}$. Let $c(D_i, t)$, $r(D_i, t)$ and $s(D_i, t)$ be the device D_i's *life clock*, *repair clock* and *subcomponent failures* at time t. The life clock is 0 when the device is new and increases with t until the device fails. The repair clock is undefined until the device fails and increases with t until the device is repaired (or replaced with a new device). If a device is operational (not under repair) at time t, then subcomponent failures (i.e. latent sector failures) are indexed by $s(D_i, t)$. For example, if sectors 234 and 425 are in error at time t, then $s(D_i, t) = \{$(i, 234),(i, 425)$\}$ (if no sector failures exist at time t, then $s(D_i, t) = \emptyset$).

The concurrent failure of components may lead to a data loss event. Let $\mathrm{DL}(F_D(t), F_S(t)) = \{0, 1\}$ be an indicator function for data loss at time t, where $F_D(t)$ are the failed components at time t and $F_S(t)$ are the subcomponent failures at time t. If $\mathrm{DL}(F_D(t), F_S(t))$ evaluates to 1, then the system has reached a failed state, otherwise, the system is still operational.

Definition 4.1.1. ***Reliability***, *$R(T_M)$, is defined as the probability that $DL(F_D(t), F_S(t))$ is 0 for all $t \leq T_M$, where T_M is the mission time of a system. In other words, it is the probability that no data loss events occur during the mission time of a system.* ***Unreliability***, *$U(T_M) = 1 - R(T_M)$, is defined as the probability that a data loss event occurs during the mission time of a system.*

Mathematically, reliability is defined as $R(T_M) = P(T_F > T_M) = 1 - P(T_F \leq T_M)$. That is, for a given mission time T_M, the probability that the system does not reach a failed state before time T_M. It follows that he unreliability is $U(T_M) = 1 - P(T_F \leq T_M)$.

Traditional storage system reliability analysis uses a metric called the mean time to data loss (MTTDL). MTTDL does not directly measure reliability; it is an expectation based on reliability: MTTDL $= \int_0^\infty R(t)dt$. This makes the MTTDL useful for relative comparisons, but awkward for absolute measurements. For example, an MTTDL measurement of 1000 years tells us very little about the probability of failure during a realistic system mission time. A system designer may be interested in the probability of failure every year for the first 10 years of a system. The aggregate nature of the MTTDL would give very little information on such a calculation.

4.2 Using Markov Models to Evaluate System Reliability

Markov models (continuous-time Markov chains) are typically used to evaluate reliability in storage systems. On one hand, Markov models allow system designers easily calculate transient measures, such as system unreliability, and steady-state measures, such as MTTDL. On the other hand, Markov models are memoryless and are only suited for modeling simple systems. In this section, we show properties of the simple, 1-disk fault-tolerant model and argue that extending this model to capture complex behavior is problematic. Complexity is added in terms of increased fault-tolerance, time dependence and irregular fault-tolerance.

4.2.1 Canonical Markov Model

The canonical Markov model used in storage systems is based on RAID 4, which can tolerate exactly one device failure. Figure 4.1 shows this model. There are a total of three states: state 0 is the state with all devices operational, state 1 is the state of one failed device and state 2 is the state where two devices have failed. Each state has at least one in/out transition (edge) with an associated rate. The sum of the outgoing transitions represent the holding time in a state. Since each transition is exponentially distributed, the holding time is also exponential. The model in Figure 4.1 has two rate parameters: failure rate (λ) and a repair rate (μ); thus, it is assumed that all devices fail at the same rate and repair at the same rate.

Starting at $t = 0$, the system is good as new and is in state 0 and remains in state 0 for an average of $(n \cdot \lambda)^{-1}$ hours (n device failures are exponentially distributed with failure rate λ), where it transitions to state 1. The system is then in state 1 for on average $(((n-1) \cdot \lambda)) + \mu)^{-1}$ hours. The system transitions out of state 1 to state 2

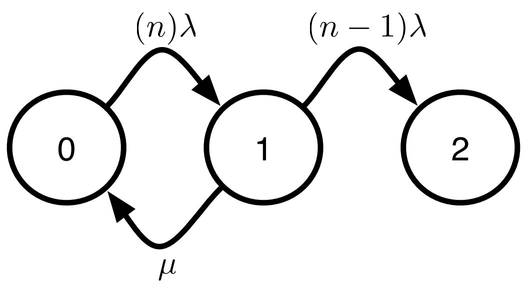

$$\begin{tikzcd}
0 \arrow[r, bend left, "(n)\lambda"] & 1 \arrow[l, bend left, "\mu"] \arrow[r, bend left, "(n-1)\lambda"] & 2
\end{tikzcd}$$

Figure 4.1: Traditional RAID 4 Markov model.

with probability $\frac{(n-1)\cdot\lambda}{((n-1)\cdot\lambda))+\mu}$, which is the failed state. Otherwise, the system transitions back to state 0, where the system is fully operational and devoid of failures.

The memoryless assumption made in these models may affect reliability analysis of a real system. Every time the system transitions back to state 0, all of the components are assumed to be good-as-new. Such a model is called *regenerative*, since the state where all components are operational (state 0) represents a brand-new system. In reality, only the recently repaired component is brand-new, while all others have a non-zero age.

4.2.2 *m*-DFT Models

Now we generalize the model in Figure 4.1 to an m-disk fault tolerant system. Figure 4.2 shows three common Markov models used to evaluate the reliability of m-disk fault-tolerant storage systems [25, 8, 19]. As with the 1-disk fault-tolerant model, the label of each state represents the current number of failures. Similarly, left-to-right transitions correspond to device failures and right-to-left transitions are repairs. The failure models within each of the three schemes are the same, while the rebuild models differ. Each model assumes one of two possible rebuild models: *serial* or *concurrent* (called parallel rebuild in [25]). The serial rebuild model assumes that at most *one* disk within a array will be rebuilt at a time. Concurrent rebuild assumes that failed disks may be rebuilt in parallel.

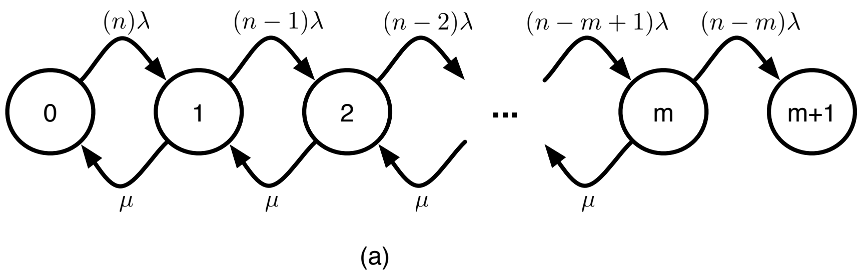
Figures 4.2-(a) and 4.2-(b) were proposed by Hafner and Rao [25] and represent serial and concurrent rebuild, respectively. For single disk fault tolerant systems, the serial and concurrent rebuild models are identical, and are correct. For multi-disk fault tolerant systems, both rebuild models are incorrect. The same modeling error leads to the errors in each model: the rebuild transitions for states 2 through m model the rebuild of the disk that failed most recently, whereas the reliability is dominated by the rebuild of the disk that failed earliest. In essence, each Markov model *resets* the rebuild time for all disks being rebuilt *whenever* another disk fails during rebuild. Thus, the model of serial rebuild models a rebuild policy that rebuilds the most recently failed disk, and resets the progress of any disks currently being rebuilt. And, the model of concurrent rebuild models a rebuild policy that restarts the rebuild of all failed disks each time a disk fails. This error highlights a systematic problem with such modeling techniques: each transition forgets about progress that has been made in a previous state.

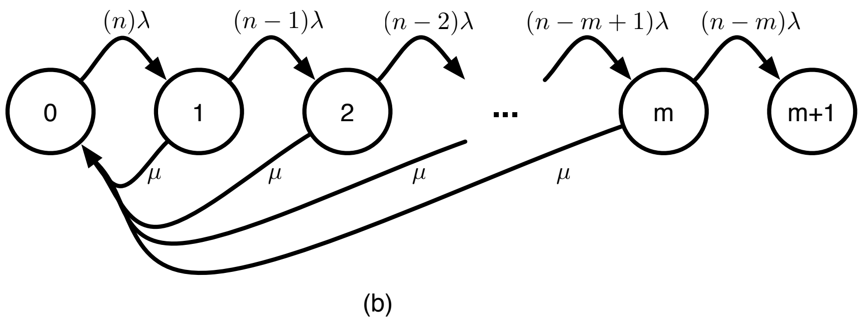
The model shown in Figure 4.2-(c) ([8, 19]) has a concurrent rebuild policy, where the rebuild transitions are very much like the failure transitions. If the system is in state i, then there are i ongoing rebuilds and the total repair rate is $i \cdot \mu$.

Resetting rebuild progress exists in all three models and can be explained as follows. Assume a repair has been ongoing for 30 hours, then the probability that the repair will last another 10 hours is

$$\begin{aligned} P(T > 30 + 10 | T > 30) &= \frac{1 - P(T \leq 40)}{1 - P(T \leq 30)} \\ &= \frac{-\mathrm{e}^{(-40\mu)}}{-\mathrm{e}^{(-30\mu)}} \\ &= P(T > 10) \end{aligned}$$

This means that the probability of a repair operation completing at time 10 and time 110 is essentially the same—the rebuild clock is totally ignored. It is known that most disk rebuilds take at least 6-10 hours and will most likely not last more than a factor of 2 or 3 beyond that. One could estimate the the remaining repair time at each state by steadily increasing the rate of ongoing rebuild operations as more failures occur. Unfortunately, as we have found, such approximations lead to unnecessary complications in the model.





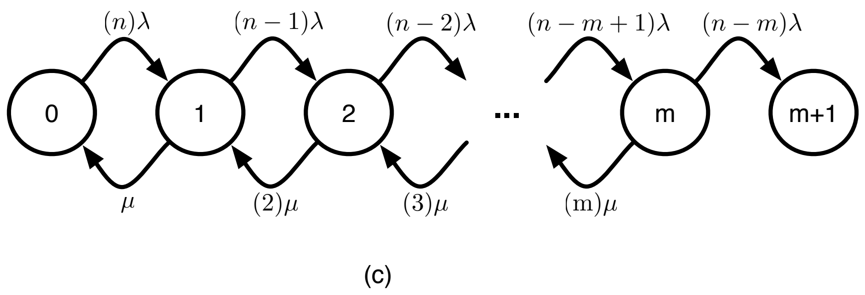


Figure 4.2: Traditional m-disk fault-tolerant Markov model.

4.2.3 Time Independence

In this section we reinforce the statements made in Section 4.2 for both failures and repairs. In addition, we show that adding time dependency in an analytical model is, in general, very difficult. We calculate the unreliability function of the analytic model in Figure 4.2-(a) to illustrate inaccuracies and difficultly in modeling time dependence.

The state residence probabilities, at time instant $t + dt$, in a system modeled by Figure 4.2-(a) are

$$\begin{aligned}
p_0(t+dt) &= p_0(t) \cdot (1 - n \cdot \lambda dt) \\
&+ p_1(t) \cdot \mu dt \\
&+ 0 \\
&\ldots \\
p_i(t+dt) &= p_i(t) \cdot (1 - (\mu + (n-i) \cdot \lambda)\, dt) \\
&+ p_{i+1}(t) \cdot \mu dt \\
&+ p_{i-1}(t) \cdot (n - i + 1) \cdot \lambda dt \\
&\ldots \\
p_m(t+dt) &= p_m(t) \cdot (1 - ((n-m) \cdot \lambda + \mu)\, dt) \\
&+ 0 \\
&+ p_{m-1}(t) \cdot (n - m + 1) \cdot \lambda dt \\
p_{m+1}(t+dt) &= p_m(t) \cdot (n-m) \cdot \lambda dt \\
&+ p_{m+1}(t) \\
&+ 0
\end{aligned}$$

where dt is assumed to be an infinitesimally small time interval, λ is the device failure rate and μ is the device repair rate. Each of these probabilities can be interpreted as follows. The probability that the system is in state i at time $t + dt$ is the sum of:

- The conditional probability that the system does not transition out of state i in the interval $[t, t + dt]$, given it was in state i at time t.

- The conditional probability that the system transitions out of state $i+1$ in the interval $[t, t+dt]$, given it was in state $i+1$ at time t.
- The conditional probability that the system transitions out of state $i-1$ in the interval $[t, t+dt]$, given it was in state $i-1$ at time t.

First observe the constant failure and repair rates: their contribution to state residence probability does not change with time. Again, this generalization reinforces the statements made in the last subsection. In short, the wear-out of a component or the progress made by a device repair is not captured by this type of model.

The instantaneous probabilities are obtained by differentiating each $p_i(t)$

$$p_i'(t) = \lim_{dt \to 0} \frac{p_i(t+dt) - p_i(t)}{dt}$$

which results in the Kolmogorov equations

$$\begin{aligned}
p_0'(t) &= p_1(t) \cdot \mu - p_0(t) \cdot n\lambda \\
&\dots \\
p_i'(t) &= p_{i+1}(t) \cdot \mu + p_{i-1}(t) \cdot (n-i+1)\lambda - p_i(t) \cdot (\mu + (n-i)\lambda) \\
&\dots \\
p_m'(t) &= p_{m-1}(t) \cdot (n-m+1)\lambda - p_m(t) \cdot (\mu + (n-m)\lambda) \\
p_{m+1}'(t) &= p_m(t) \cdot (n-m)\lambda
\end{aligned}$$

Using a Laplace transform or numerical methods, we solve the system of differential equations and compute the system unreliability as

$$U(t) = p_{m+1}(t).$$

Solving the Kolmogorov equations represents the standard method for evaluating the reliability of a fault-tolerant storage system. We use the same method throughout our analysis to evaluate the unreliability function for Markov models.

As illustrated in the derivation, all transition rates remain constant in time. Again, this is the fundamental building block of these models. These models stay quite simple as long as all rates are constant, even if the individual failure/repair rates are different. Since the rates remain constant (independent of time), they can simply be added together to obtain total state in/outgoing rates. In reality, these rates *may be* time dependent, resulting in rates that vary with time. In this case, the summation of rates becomes quite difficult. In addition, individual device lifetime clocks and repair clocks makes a similar analytical solution extremely difficult.

To illustrate this difficulty, recall the life clock ($c(D_j, t)$) and repair clock ($r(D_j, t)$) of device j. A device's life clock begins at 0 when the device is added to the system (via new installation or repair), increases with time and expires when the device fails. A device's repair clock begins at 0 when the device fails, increases with time and expires when the repair is complete. When a clock expires, it evaluates to $\emptyset$. Note that, while all of the clocks take the global system time, t, as a parameter, each clock value may be different.

If we assume time-dependent failure and repair distributions, each device must be considered individually in the state residence probabilities. Let $\lambda_j(t)$ and $\mu_j(t)$ be the hazard function of the j-th device's failure and repair distributions, respectively. Also, let $\lambda_j(\emptyset) = 0$ and $\mu_j(\emptyset) = 0$. The failure rate and repair rate of device j at time t is $\lambda_j(c(D_j, t))$ and $\mu_j(r(D_j, t))$, respectively. In this case, the failure transition rate at time t is

$$\left(\sum_{a \in A} \lambda_a(c(D_a, t))\right)$$

where A indexes the current available devices. Similarly, the repair transition rate at time t is

$$\left(\sum_{f \in F} \mu_f(r(D_f, t))\right)$$

where F indexes the current failed devices.

Correctly incorporating these time-dependent failure and repair transitions into the state residence probabilities is quite difficult and in many cases may not be possible. The difficulty lies in the distinction between *absolute time* and *relative time.* We consider absolute time to be the time since the system was generated, while relative

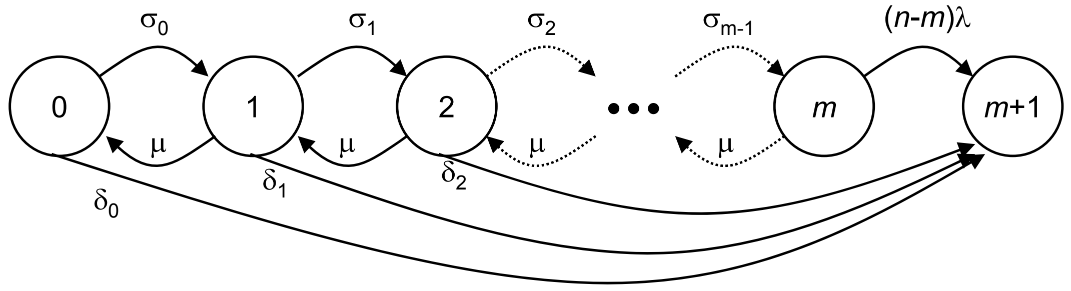


Figure 4.3: Markov model for irregular erasure code (cf. Figure 2 in [25]).

time applies to the individual device lifetime and repair clocks. Analytic models operate in absolute time; therefore, there is no reasonable way to determine the values of each individual clock. For this reason, we believe that the only way to effectively evaluate the reliability of a system with time-dependent failures and repairs is through simulation.

4.2.4 Modeling Irregular Codes

One key aspect of this work involves understanding the fault tolerance and reliability of irregular erasure codes. The concerns we have discussed thus far regarding Markov models of disk rebuild and sector failures applies to erasure codes in general, whether they be regular or irregular in nature. However, the nature of irregular erasure codes exacerbates the difficulty in accurately modeling such storage systems.

Hafner and Rao have used Markov models to analyze the reliability of irregular codes [25]. Although they include BER sector failures from a critical mode in their model, to simplify discussion, we do not. We reproduce pertinent aspects of their Markov model in Figure 4.3.

Like the other Markov models, the transition rates from left to right corresponds to disk failures, and the ones from right to left correspond to disk rebuilds. This Markov model is distinguished from the previous models in that two transitions leave each disk failure state: a “normal” one that transitions to the next failure state, and another that transitions directly to the data loss state $m + 1$. In a Markov model for a serial rebuild policy of a regular code, the transition rate σ_i is $(n - i)\lambda$ (cf. Figure 4.2). For an irregular code, the fault tolerance vector—or it’s complement, the fault tolerance matrix (explained in Section 5.3.2) —is used to assign a likelihood that the irregular code tolerates a single disk failure. If f_i is the i^{th} entry of the fault tolerance vector, then the transition rates in Figure 4.3 are $\sigma_i = (1 - f_i)(n - i)\lambda$ and $\delta_i = f_i(n - i)\lambda$.

The Markov model shown in Figure 4.3 provides good intuition about how to think about the reliability of irregular codes. Unfortunately, revising the Markov model for irregular codes to accurately model rebuild policies, and to include sector failures appears to be impractical and is prone to error.

Including either latent or BER sector failures in a Markov model of an irregular erasure code is difficult. Irregularity does not seem to complicate the inclusion of latent sector failures and scrubbing any more than an MDS code. Though recall that we reached the conclusion that including latent sector failures and scrubbing in a Markov model of a regular code was too complicated. Hafner and Rao included BER sector failures in their Markov model [25]. As is the case for MDS erasure codes, the BER failure probability must be a function of the portion of disk that is critically exposed, and that is difficult to capture in a Markov model.

Many irregular erasure codes are HoVer codes [22]. This adds additional complexity to the modeling of BER sector failures. In HoVer codes, the determination of which sectors are critically exposed given a combination of other disk and sector failures is complicated. We believe that the method that Hafner and Rao use to determine this probability (cf. Equation 3.2 in [25]), which involves entries i and $i + 1$ in the fault tolerance vector, only applies to flat irregular codes. To be more concrete, consider the Weaver [21] codes that they analyze. Weaver codes have a symmetric structure and have both data and parity elements in each strip. The loss of a specific additional disk (strip) leads to data loss does not mean that the loss of any element in the strip leads to data loss. For example, if one of the parity elements in the strip is necessary to rebuild a given set of failed disks, but none of the data elements are, then the fault tolerance vector does not provide sufficient information to estimate the likelihood of a BER failure. The fact that only some elements in the strip are necessary needs to be taken into account when determining the probability that a BER causes data loss. We have developed a structure, called the *fault tolerance matrix*, which accounts for both strip and partial strip errors. The fault tolerance matrix is discussed in section 5.3.2.

4.3 Modeling Sector Failures

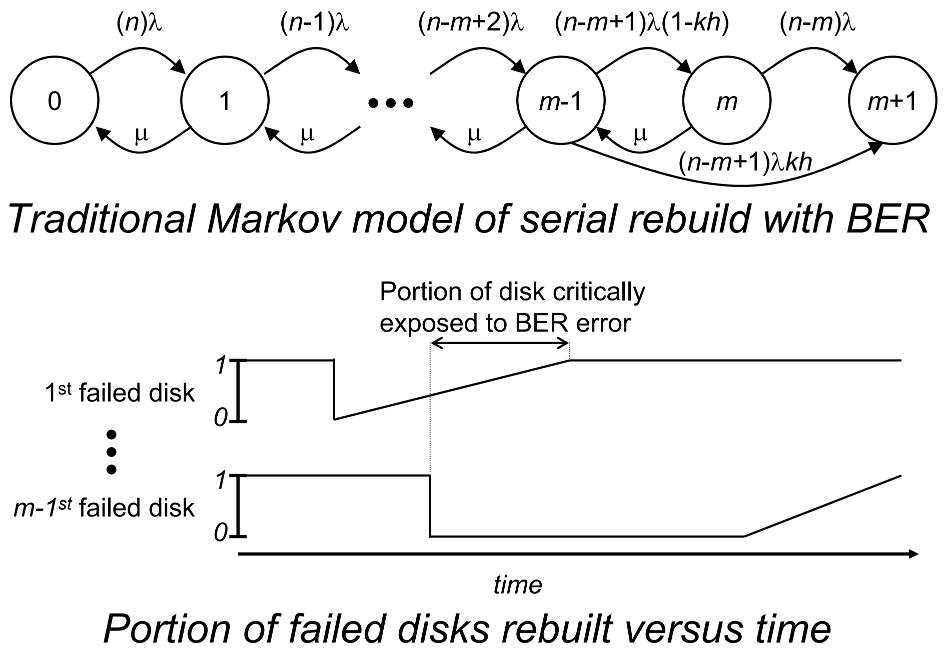
The storage community has not reached consensus on how best to model sector failures. There are currently two distinct approaches to modeling sector failures in storage systems. In the first approach, a bit error rate (BER) is assumed and is used to model data loss when the system is performing a rebuild in critical mode. In a m DFT system, the model enters *critical mode* upon the m-th disk failure. In the second approach, latent sector failures are assumed to appear at some rate. The rate at which latent sector failures are assumed to occur is based on time, usage, age, historical data, and so on. Storage systems are assumed to have a scrubbing policy that finds and recovers latent sector failures before a critical stage is entered.

We believe that a detailed model of storage system reliability may have to account for both BER during the critical mode, and latent sector failures that develop over time. Both types of sector failures are difficult to correctly model and accurately include in traditional Markov models.

With the exception of Elerath and Pecht's simulation [14], most sector failure models are built as an extension of the canonical Markov model. That is, additional failure transitions are added into a Markov model to account for sector failures. As we will show, the critically exposed region cannot be fully captured when augmenting the canonical Markov model with sector failure transitions. Elerath and Pecht [14] treat sector failure and scrub transitions individually, which the increases complexity of augmented Markov models and simulations. We propose a method for drawing sector errors that can be applied to either a BER or latent sector error model.

4.3.1 BER Failures

Consider modeling the effect of sector failures during rebuild from critical stage. The Markov model in Figure 4.4 is a traditional Markov model of a serial rebuild policy augmented to include data loss due to BER. The figure is based on the one developed by Hafner and Rao [25]. Remember that $n = k + m$, and so the term $n - m + 1$ in the transition to state m can be written as $k + 1$. The parameter h is the BER multiplied by the capacity of the device; i.e., the likelihood that a single disk exhibits a bit error if read in its entirety. Multiplying the term h by k accounts for the number of disks that



(n)λ
(n-1)λ
(n-m+2)λ
(n-m+1)λ(1-kh)
(n-m)λ
0
1
m-1
m
m+1
μ
(n-m+1)λkh
Traditional Markov model of serial rebuild with BER
Portion of disk critically exposed to BER error
1st failed disk
m-1st failed disk
1
0
time
Portion of failed disks rebuilt versus time

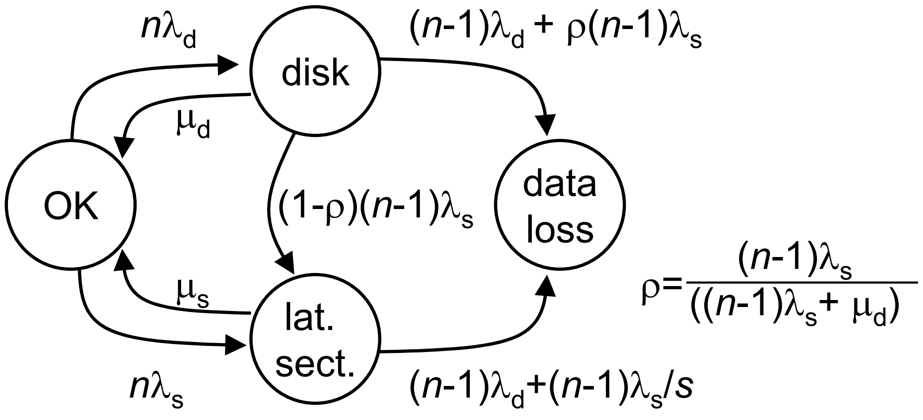
Figure 4.4: BER sector failure Markov model.

must be read to rebuild a disk from this state.

We find three simplifying assumptions in this model. First, regardless of whether a sector failure is due to BER or developed latently, it only leads to data loss if it occurs in the portion of the failed disk that is critically exposed. For example, in a two disk fault tolerant system, if the first disk to fail is 90% rebuilt when a second disk fails, only 10% of the disk is critically exposed. Traditional Markov models that use the BER sector failure model do not accounted for the critically exposed region of a rebuild process.

Beneath the Markov model is an illustration of the *critically exposed* portion of a disk. Given a serial rebuild policy, some portion of the first disk to fail will be rebuilt by the time the $m - 1^{\text{st}}$ disk fails. The rebuilt portion of the first failed disk can be used if any of the remaining k disks have a sector failure in that block range. Thus, only the portion of the first failed disk that has not yet been rebuilt is critically exposed on the $n - m = k$ other available disks.

The second problematic assumption is in the likelihood calculated using the



$n\lambda_d$
disk
$(n\text{-}1)\lambda_d + \rho(n\text{-}1)\lambda_s$
μ_d
OK
$(1\text{-}\rho)(n\text{-}1)\lambda_s$
data loss
$\rho = \frac{(n\text{-}1)\lambda_s}{((n\text{-}1)\lambda_s + \mu_d)}$
μ_s
lat. sect.
$n\lambda_s$
$(n\text{-}1)\lambda_d + (n\text{-}1)\lambda_s/s$

Figure 4.5: Latent sector failure Markov model for a 1-DFT system. This model is based on Elerath and Pecht's approach.

BER approach. It is assumed that the likelihood is bound by 0 and 1: $0 \leq h \leq 1$. In the original calculation, it is possible for h to exceed 1. Suppose that the BER is 1×10^{-12} and the device capacity is 8×10^{12} bits. In this case the likelihood is 8.0, which is not a valid probability. The actual calculation should be $h = 1 - (1 - \text{BER})^C$, where C is the device capacity in bits. In the case of our example, the value of h will be 0.9996, instead of 8.0. The $h \cdot k$ calculation suffers from the same problem. Instead of $h \cdot k$, the calculation should be $P_{\text{BER}} = 1 - (1 - h)^k$.

Finally, this approach does not compute the likelihood that 1, 2, ... sector errors exist. It simply computes the probability that one or more errors exist. Such a calculation is especially important when considering the critically exposed region.

4.3.2 Latent sector failures

Some systems model latent sector failures and the scrubbing process that detects and repairs them. There are two ways to model these processes: treat sector failures (scrubs) as most models treat disk failures (repairs) [14] or model sector failure/scrubs as a separate random process [56, 29].

Elerath and Pecht [14] included latent sector errors and scrubbing as explicitly separate rates into their model for a single-disk fault tolerant system. Elerath and Pecht calculated a sector failure rate from the BER and an assumed workload. The scrub rate was chosen at a week (168 hours). Once the disk was scrubbed all latent sector errors were assumed repaired as long as no disks were failed. Applying this approach to the canonical 1-disk fault-tolerant Markov model results in the model shown in Figure 4.5. As we see, adding latent sector faults—via sector failure and scrub transitions—to a simple Markov model is quite complicated; the complexity quickly grows as the fault-tolerance is increased.

Iliadis *et al.* proposed a model that combines the latent sector failure rate and scrub rate into a single probability [29]. Schwarz *et al.* derived a similar model. Here, we focus on a deterministic scrub schedule. Given a deterministic scrub period for a sector, T_S, load on a given sector, l, probability of a sector error due to a write operation, P_w and the ratio of write requests in the total system load, r_w, the probability of an unrecoverable error on a given sector at an arbitrary time is [29]

$$P_S = \left(1 - \frac{1 - e^{-lT_S}}{lT_S}\right) \cdot P_w \cdot r_w.$$

This probability was derived for use in a Markov model, requiring it to be independent of the underlying device age. Even though there is not enough data available to create a time-dependent sector failure distribution, a time-dependent probability can be derived as follows. Iliadis *et al.* derived

$$P_S(t) = \left(1 - e^{-l(t \bmod T_S)}\right) P_e,$$

where P_e is the single sector failure probability. If we take t to be the current device age, $F(t)$ to be the cumulative probability distribution function and assume that the device is scrubbed every T_S hours, then

$$P_S(t) = \left(F(t) - F\left(T_S \cdot \left\lfloor \frac{t}{T_S} \right\rfloor\right)\right) \cdot P_e$$

will give the time-dependent sector failure rate. Incorporating load into such a calculation and determining a proper sector failure model remains an open problem.

P_S can be used to calculate the likelihood of a latent sector failure on a disk. If a disk has C sectors, then the probability that one or more latent sector errors are present at an arbitrary point in time is

$$P_{LS} = 1 - \Pr(\text{no errors in } C \text{ sectors}) = 1 - (1 - P_S)^C.$$

This probability may be used in place of P_{BER} during the critical stage to estimate the effect of latent sector errors on reliability. Unfortunately, when used in a Markov model, such a calculation suffers the same problem as the BER method: it does not account for the critically exposed region of the rebuild process or the number of sector errors.

4.3.3 Critical Region Sector Error Model

There are three major issues with previous sector error models. First, none of the models account for the critical region of the rebuild process. Second, Elerath and Pecht's model works quite well for the 1-disk fault tolerant case, but becomes extremely complicated and does not account for the critical region in multi-disk fault-tolerant systems. Finally, no model accounts for the number and location of sector errors. Here we propose a model based on Iliadis *et al.* that accounts for all of these shortcomings.

Recall the probability of a latent sector error on a disk with C sectors:

$$P_{LS} = 1 - \Pr(\text{no errors in } C \text{ sectors}) = 1 - (1 - P_S)^C.$$

An approximation to account for the portion of disk that is critically exposed is calculated by assuming the subsequent disk failures are uniformly distributed during rebuild time. Consider the multi-disk fault-tolerant model shown in Figure 4.2. The transition from state with $m-1$ disk failures to state with m disk failures has rate $(n-m+1)\lambda(1 - P_{\mathrm{LS}}2^{-(m-1)})$. Another transition is added from the state with $m-1$ disk failures to the data loss state: $(n-m+1)\lambda(P_{\mathrm{LS}}2^{-(m-1)})$. Dividing by the term 2^{m-1} approximates the portion of the first failed disk that is critically exposed to BER failure.

If simulation is used to estimate storage system reliability, then a more accurate method of determining sector errors in the critical region may be used. For each failed device, calculate the number sectors that have yet to be rebuilt, C'. We approximate the number of sector errors by first drawing to determine if there is at least one latent sector error and draw again to determine exactly how many errors exist by computing the following conditional probabilities. The probability that there is exactly $i > 0$ latent sector errors, given there is at least one latent sector error is

$$\begin{aligned}\Pr\left(\text{exactly } i \text{ LS errors} \mid \text{at least one LS error}\right) &= \frac{\Pr\left(\text{exactly } i \text{ LS errors}\right)}{\Pr\left(\text{at least one LS error}\right)} \\ &= \binom{S}{i} \cdot \frac{\left(P_S\right)^i \cdot \left(1 - P_S\right)^{C'-i}}{1 - \left(1 - P_S\right)^{C'}}\end{aligned}$$

The error locations may be drawn from distributions based on error locality, or uniformly across the device. Once the error locations are drawn for each device, the simulation can determine if a data loss event has occurred.

4.3.4 Modeling Sector Errors: Discussion

While explicitly including sector failures and scrubbing as separate rates works quite well for 1-DFT systems, modeling sector failures at this granularity for multi-disk fault-tolerant systems will over complicate an analytical model and consume unnecessary cycles in simulation. The only reason to do such fine-grained analysis is when the probability of two or more sector errors in the same stripe is comparable to two concurrent disk failures when there are $m - 1$ ongoing repairs in an m-disk fault-tolerant system. Assuming a stripe size of N and C sectors per disk, the (estimated) probability of two or more sector errors in the same stripe is

$$P_{LS} \cdot \left(1 - \left(1 - P_S\right)^{N-1}\right).$$

Note that this estimate is quite optimistic; it ignores the critical region of the rebuild process. This is interpreted as the probability that a latent sector error is on an arbitrary disk and there exist one or more errors in the same stripe as the initiating error. For an array of eight 300 GB disks, when $P_S = 2.36 \times 10^{-11}$, this probability is 2.27×10^{-12}. In otherwords, highly unlikely. In fact, under most conditions in an 8 disk array, this is larger than the probability of four concurrent disk failures (cf. Section 6.2).

Since sector errors are expected to occur much more often than disk failures, explicitly keeping track of individual sector failures in multi-disk fault-tolerant simulations over complicate the simulation and consumes unnecessary cycles with very little return. Additionally, outside of single-disk fault-tolerant systems, explicitly modeling multiple sector errors in a single stripe will lead to a state-space explosion. To make matters worse, most of the new paths added to the model will be extremely low probability paths.

The jury is still out on how the BER and latent sector models are related. It is possible that the two models will be combined. In fact, without scrubbing, the model presented by Iliadis *et al.* is the BER model. Until the probability of multiple sector failures in a stripe approaches that of two concurrent disk failures, modeling sector failures using P_S during the critical stage appears to be the best option.

4.4 Reliability Modeling: Discussion

Here we have analyzed the traditional reliability techniques used to model storage systems. In doing so, we have encountered the following issues with current modeling techniques:

Memorylessness: The underlying assumption of Markov models assumes time-independence. This makes accurate failure and rebuild modeling very difficult.

Critical Region: The sector failure aspect of current analytic models do not account for the critical rebuild region when determining data loss events. The models assume the entire device is the critical region, leading to an overestimate in system unreliability.

Many Paths, Little Utility: The fine-grained latent sector model is sufficient for 1 DFT systems, but results in a state-space explosion as fault tolerance is increased. The state residence probabilities of most of the newly added states are extremely low, resulting in little utility for a great increase in complexity.

Aggregation of Irregularity: If an irregular code is to be modeled by a Markov model, the data loss probabilities must be aggregated into a single vector (or matrix). In the case of HoVer codes, such aggregation may lead to poor estimates.

Another aspect not yet mentioned is the expected amount of data loss. No current model has incorporated such a metric. While an accurate estimate on the expected number of bytes lost during a data loss event remains open, we propose the following approximation over N iterations of a simulator

$$\text{Avg. Data Loss} = \frac{\sum_{i=1}^{N} I(T_F < T_M) \cdot S_E}{N}$$

where S_E represents the critically exposed sectors during a data loss event. If a data loss event was triggered by a double disk failure, then this value would equal the capacity of an entire device. If the data loss event was triggered by a set of disk failures plus a sector failure, the value is a single sector (512 bytes).

Modeling the reliability of irregular erasure codes is more complicated than modeling the reliability of MDS erasure codes. Unfortunately, all of our concerns about the accuracy of Markov models for multi-disk fault tolerant erasure codes and the inclusion of sector failures are magnified by irregularity. For these reasons, we believe that simulation is the best way to understand reliability tradeoffs between different erasure codes. Unfortunately, such simulations do not yield elegant closed-form solutions from which an intuitive understanding of the impact technology trends in storage capacity, MTTF, MTTR, and so on will have on system reliability.

Chapter 5

Reliability Simulation of Erasure-coded Storage Systems

Chapter 4 was an exploration of traditional techniques for modeling reliable storage systems. We found that Markov models are not do not sufficiently model systems with time dependence, multi-disk fault tolerance, irregular fault-tolerance and sector errors. As a result, the only general way to effectively model complex, erasure-coded storage systems is simulation.

In this chapter, we review standard simulation techniques, which lends itself to very flexible configurations. While standard simulation readily handles time dependence, there are cases where effectively determining data loss events and obtaining a result in a reasonable amount of time is non-trivial. We have developed novel bookkeeping mechanisms to efficiently determine data loss events in the face of both disk and sector failures. As the fault-tolerance of a storage system increases, the running time of standard simulation increases at a rapid rate. We utilize fast simulation techniques to drastically speed up simulation runs of highly-fault tolerant systems.

We begin this chapter by setting up relevant terminology and standard simulation techniques. Next, we describe the main drawback of standard simulation: *rare events*. A rare event is an event that is highly improbable and will most likely not be observed in standard simulation. A data loss event in a highly fault-tolerant system is an example of a rare event. We describe two fast simulation techniques that are used to increase the probability of data loss events in simulation.

Finally, we describe the architecture of the High-Fidelity Reliability (HFR) Simulator. The bookkeeping methods and fast simulation techniques are implemented in the HFR Simulator. Given an arbitrary linear erasure code and a system configuration, the HFR Simulator has the ability to efficiently and accurately provide a reliability estimate. The bookkeeping methods leverage prior work on *minimal erasures* [65] to achieve *high-fidelity* simulation, which is necessary to simulate irregular erasure codes. Minimal erasures concisely and precisely describe the fault tolerance of an irregular erasure code. We use minimal erasures in the HFR Simulator to efficiently perform the bookkeeping necessary to determine if some set of disk and sector failures leads to data loss. The HFR Simulator may also use a coarse-grained method of bookkeeping based on the *fault tolerance matrix*, a pre-computed table of failure probabilities given the number of disk and sector failures; this approach is more efficient than minimal erasures bookkeeping, but only appropriate for some irregular erasure codes.

5.1 Standard Simulation of Erasure-Coded Storage Systems

The objects in our simulation framework are storage devices. There are exactly n devices, which may constitute an array: $D_0, D_1, \dots, D_{n-1}$. Each device has two clocks that determine the device's age and rebuild progress. Let $c(D_i, t)$ and $r(D_i, t)$ be device D_i's *life clock* and *repair clock* at time t. The life clock is 0 when the device is new and increases with t until the device fails. The repair clock is undefined until the device fails and increases with t until the device is repaired (or replaced with a new device). If a device is operational (not under repair) at time t, then subcomponent failures (i.e. latent sector failures) are indexed by $s(D_i, t)$. For example, if sectors 234 and 425 are in error at time t, then $s(D_i, t) = \{$(i, 234),(i, 425)$\}$ (if no sector failures exist at time t, then $s(D_i, t) = \emptyset$). Under this framework, any of the sector failure methods explained in Section 4.3 may be used to determine $s(D_i, t)$.

Each device, D_i, has a failure distribution with CDF $F_i(t)$ and a repair distribution with CDF $R_i(t)$. The failure distribution has hazard function $\lambda_i(t)$, while the repair distribution has hazard function $\mu_i(t)$. We can draw random variates from each distribution using either the corresponding CDF or cumulative hazard function. Given a random variate drawn from the uniform distribution on the unit interval $(0,1)$, U, we can use *inverse transform sampling* [43] to draw a time, T, from the CDF $F_i(t)$

$$T = F_i^{-1}(U).$$

The cumulative hazard function represents the accumulation of the hazard rate over time and is calculated as $\Lambda_i(t) = \int\limits_0^t \lambda_i(x)dx$. It turns out that

$$F_i(t) = 1 - \exp\left(-\Lambda_i(t)\right),$$

allowing us to use the inverse transform method to sample using the cumulative hazard function. This is especially useful when sampling failure times from a truncated distribution. If the current age of a device is a, then we can sample the failure time from

$$1 - \exp\left(-\int\limits_0^t \lambda_i(a+x)dx\right).$$

The available and failed devices are indexed by the sets $\textsc{avail}$ and $\textsc{failed}$. The entries of $\textsc{avail}$ and $\textsc{failed}$ are simply the device indices; if device D_i is currently failed, then $i \in \textsc{failed}$. Additionally, sector failures are indexed by $\textsc{sectors}$. An entry of $\textsc{sectors}$ will contain the device index and the corresponding sector offset within the device.

Since we are measuring reliability in a fault-tolerant data storage system, the concurrent failure of components and sub-components may lead to a data loss event. Let $\textsc{dl}(\textsc{failed}, \textsc{sectors}) = \{0, 1\}$ be an indicator function for data loss given the current set of failed devices and sectors. If $\textsc{dl}(\textsc{failed}, \textsc{sectors})$ evaluates to 1, then the system has reached a failed state, otherwise, the system is still operational. The actual bookkeeping methods used to implement the $\textsc{dl}$ function are covered in Section 5.3.3.

5.1.1 Measuring System Reliability via Simulation

At a high level there are two ways to evaluate the reliability of an erasure-coded system using simulation [38, 42, 14]. The first type of simulation estimates the system unreliability function at time t

$$U(t) = \Pr(T_F \leq t) = E\left[I(T_F \leq t)\right],$$

where $I(\cdot)$ is an indicator function and T_F is a failure time. The method of replications, which we call iterations, is used to obtain the estimator, $\hat{U}(t)$, of $U(t)$. In each iteration, the system is simulated until a failure event (i.e. data loss) or a pre-specified mission time, T, is exceeded. Given N iterations, the estimator is computed as

$$\hat{U}(t) = \frac{1}{N}\sum_{i=1}^{N} I(T_F \leq t).$$

The second type of simulation obtains the mean time to data loss (MTTDL). We also use an iterative method to estimate the MTTDL. At each iteration, the system is simulated until a data loss event. Given N iterations, the MTTDL is estimated by

$$\text{MTTDL} = \frac{1}{N}\sum_{i=1}^{N} T_i,$$

where T_i is the stopping time of the i-th iteration. This is the method of simulation we use in Chapter 7. In general, this particular method is fairly inefficient (even for 1-DFT systems). The method was used to match the RME metric developed in Chapter 7.

The accuracy of a set of simulated measurements is determined by confidence intervals and relative error [38, 42]. Assume iteration i returns the value x_i and the mean over all iterations is x_M. We compute the standard deviation as

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - x_M)^2}.$$

The standard deviation is then used to construct a 90 percent confidence interval with endpoints $x_M - 1.645\frac{s}{\sqrt{n}}$ and $x_M + 1.645\frac{s}{\sqrt{n}}$. The metric we report in our results is the relative error, which is calculated as $\frac{1}{x_M}1.645\frac{s}{\sqrt{n}}$.

As we will show in Section 5.1.2 both of the aforementioned simulation methods are inefficient when evaluating highly fault-tolerant systems. Fortunately, the method of computing system unreliability lends itself to efficient simulation of highly fault-tolerant systems. In addition, under appropriate technical conditions, a similar method may also be used to estimate MTTDL [42].

Algorithm 1 The outer loop of a reliability simulator

```
$N \leftarrow$ number of iterations
$S \leftarrow 0$
$i \leftarrow 0$
**while** $i < N$ **do**
    $SUM \leftarrow SUM +$ iteration()
**end while**
**return** $\frac{S}{N}$
```

The outer loop of the simulator is shown in Algorithm 1, where the iteration function returns either $\{0, 1\}$ in the mission time simulation, a failure time in the MTTDL simulation or $\{0, L\}$ in the methods presented in Section 5.2.

Algorithm 2 illustrates a single iteration of a mission-time simulator. An iteration of this type of simulator runs until failure or the simulation clock value exceeds a predefined value. An iteration of the simulator proceeds as follows. At time $t = 0$, the system is brand new and we draw failure times for each device from $F_i(t)$. The failure times are sorted, stored along with device index in FAILTIMES, and the first event corresponds to the minimum failure time. After the first event is processed, an event will either be a failure or repair. An event is triggered by determining the event located temporally closer to t. If the triggered event time exceeds a mission time, then the iteration evaluates to 0 and the iteration terminates. If a failure event is triggered, a repair time is scheduled (drawn) and stored in REPAIRTIMES. Next, the simulator checks to see if any sector failures have occurred on the available devices. The sector failure process includes drawing to determine where the failures are and culling the sectors that are not in a critical region (cf. Section 4.3). Finally, DL(FAILED, SECTORS) is evaluated for a data loss event. The iteration evaluates to 1 if a data loss event has occurred. If a repair event is triggered, then a new failure time is scheduled for the repaired device.

An iteration of a simulator unbounded in time is shown in Algorithm 3. The mechanics of this type of simulator are very similar to a mission-time simulation. There are two main differences. First, an iteration will run until failure, which in theory could be on the order of days, weeks or years (real-time). Finally, instead of returning 1 or 0, the actual failure time is returned.

Algorithm 2 A single iteration a mission-based discrete-event simulation.

```
1: $t \leftarrow 0$
2: $T \leftarrow$ mission time
3: AVAIL $\leftarrow \{0, 1, \ldots, n\}$
4: FAILED $\leftarrow \emptyset$
5: FAILTIMES $\leftarrow \{(t_1, i_1), (t_2, i_2), \ldots, (t_n, i_n)\}$
6: REPAIRTIMES $\leftarrow \emptyset$
7: while true do
8:   $(t_F, i_F) \leftarrow \{(t', i') \in$ FAILTIMES : $t'$ is the min time over all entries$\}$
9:   $(t_R, i_R) \leftarrow \{(t', i') \in$ REPAIRTIMES : $t'$ is the min time over all entries$\}$
10:  if $\min(t_F, t_R) > T$ then
11:    return 0
12:  end if
13:  if $t_F < t_R$ then
14:    FAILED $\leftarrow$ FAILED $\bigcup \{i_F\}$
15:    AVAIL $\leftarrow$ AVAIL $\setminus \{i_F\}$
16:    SECTORS $\leftarrow$ get_failed_sectors(AVAIL)
17:    if DL(FAILED, SECTORS) == 1 then
18:      return 1
19:    end if
20:    REPAIRTIMES $\leftarrow$ REPAIRTIMES $\bigcup \left\{R_{i_F}^{-1}(U) + t_F, i_F\right\}$
21:  else
22:    FAILED $\leftarrow$ FAILED $\setminus \{i_R\}$
23:    AVAIL $\leftarrow$ AVAIL $\bigcup \{i_R\}$
24:    FAILTIMES $\leftarrow$ FAILTIMES $\bigcup \left\{F_{i_R}^{-1}(U) + t_R, i_R\right\}$
25:  end if
26: end while
```

Algorithm 3 A single iteration of an unbounded discrete-event simulation.

```
1: $t \leftarrow 0$
2: AVAIL $\leftarrow \{0, 1, \ldots, n\}$
3: FAILED $\leftarrow \emptyset$
4: FAILTIMES $\leftarrow \{(t_1, i_1), (t_2, i_2), \ldots, (t_n, i_n)\}$
5: REPAIRTIMES $\leftarrow \emptyset$
6: while 1 do
7:   $(t_F, i_F) \leftarrow \{(t', i') \in$ FAILTIMES : $t'$ is the min time over all entries$\}$
8:   $(t_R, i_R) \leftarrow \{(t', i') \in$ REPAIRTIMES : $t'$ is the min time over all entries$\}$
9:   if $t_F < t_R$ then
10:    FAILED $\leftarrow$ FAILED $\bigcup \{i_F\}$
11:    AVAIL $\leftarrow$ AVAIL $\setminus \{i_F\}$
12:    SECTORS $\leftarrow$ get_failed_sectors(AVAIL)
13:    if DL(FAILED, SECTORS) $== 1$ then
14:      return t
15:    end if
16:    REPAIRTIMES $\leftarrow$ REPAIRTIMES $\bigcup \left\{ R_{i_F}^{-1}(U) + t_f, i_F \right\}$
17:  else
18:    FAILED $\leftarrow$ FAILED $\setminus \{i_R\}$
19:    AVAIL $\leftarrow$ AVAIL $\bigcup \{i_R\}$
20:    FAILTIMES $\leftarrow$ FAILTIMES $\bigcup \left\{ F_{i_R}^{-1}(U) + t_R, i_R \right\}$
21:  end if
22: end while
```

With the exception of how we process sector failures and determine data loss events, these algorithms represent standard discrete event simulation.

5.1.2 Rare Events

So far, we have discussed standard simulation methods for fault-tolerant systems. The major drawback to this type of simulation is the amount of time required to get an accurate result when the probability of failure is extremely low. We assume that the component failure rates are orders of magnitude higher than the repair rate. In a system that is one disk fault-tolerant, standard simulation is sufficient [14]. Under the usual conditions the probability of failure in 10 years is roughly of the order 10^{-4}, thus requiring roughly 10^4 iterations to reach a data loss event (cf. Table 6.6, Section 6.2).

As the fault-tolerance of the system increases, the probability of failure decreases by orders of magnitude and many more iterations are required to get an accurate reliability measure. For instance, under the current failure/repair parameters, a 3 disk fault-tolerant system has an unreliability of about 10^{-11} over 10 years, thus a single data loss event is expected to occur after roughly 100 billion iterations of the simulator. If an iteration in an extremely efficient simulator takes 1 ms, then we will expect a single data loss event in 31 years. If we parallelize and saturate 1000 nodes, the calculation will take 11 days. Note, that this is for a single data loss event; we need many such observations to get statistically significant probabilities.

Figure 5.1 illustrates the idea of rare events in highly fault-tolerant systems. We assume that a set of devices can tolerate the failure of any m devices before losing data. As time increases, devices randomly fail according to some distribution and as a result are repaired. In general, the repair times are orders of magnitude smaller than the expected failure times. Since the system is m device fault tolerant, at least $m+1$ device repair operations must overlap in order to have a data loss event. Given a timeline that is, say 10 years, where a device may fail 1 or 2 times, observing multiple, overlapping 12 hour windows is quite rare.

As we will see, the probability of observing, say 2 overlapping device failure events is tractable using standard simulation. Simulating systems with more fault tolerance becomes increasingly difficult as fault-tolerance increases.

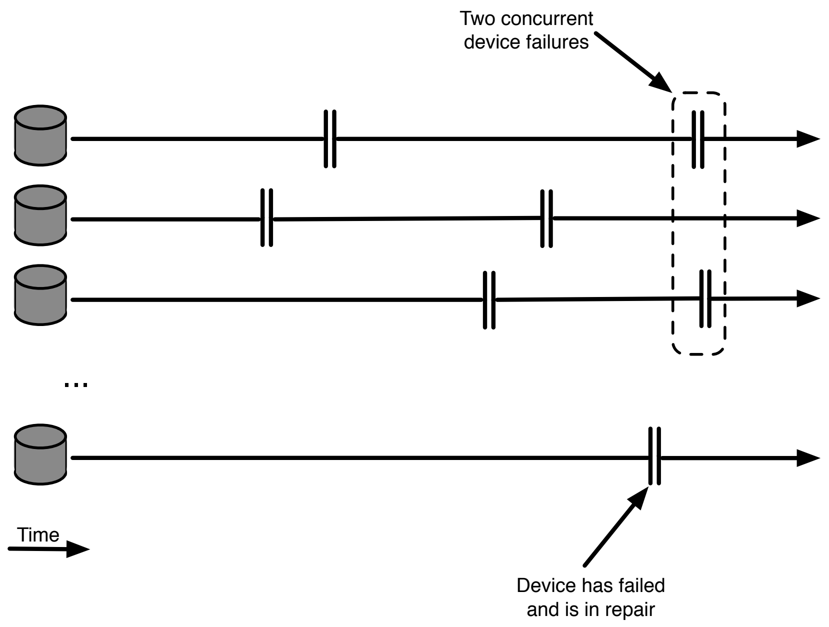

Two concurrent device failures
...
Time
Device has failed and is in repair

Figure 5.1: A timeline of an array of devices that have large failure times and very small repair lifetimes. When the time to repair is orders of magnitude smaller than the time to fail, multiple concurrent devices failures represent a *rare event*.

5.2 Increasing the Probability of Observing Rare Events

The current state-of-the-art for observing rare events in simulation is a technique called *Importance Sampling* (IS) [41, 40, 39]. Informally, IS simply accelerates the failure of subsequent device failures after a pre-defined threshold. If the system is in a non-failure state, then the simulation proceeds as a standard Monte-Carlo simulation: the device failures and repairs are drawn from their original distributions. Once the number of device failures is equal to the pre-defined threshold, subsequent failures are "accelerated" until a data loss event, or the system reaches the non-failed state. The threshold is typically chosen as a single device failure. Accelerating the additional device failures requires a change in measure in the device failure distribution. This change in measure must be carefully chosen (usually empirically) to ensure accurate estimates.

It is important to note that IS techniques are *variance reduction* techniques. That is, given a system simulated for k iterations under standard simulation and a system simulated under IS for the same number of iterations, the system under IS is expected to have a lower variance. While the variance is generally expected to be lower, there are two subtleties associated with IS techniques: there are cases where IS techniques may lead to higher variance or may not accurately model systems that have extremely low reliabilities. The simulation community has not found general guidelines for IS parameters; thus, the appropriate parameters for IS must be found empirically. Chapter 6 includes an empirical evaluation of suitable IS parameters.

Without a correction factor, IS will result in a biased estimate. The estimate is unbiased using a value called the *likelihood ratio*. Assume that $f(x)$ is the probability measure governing the dynamics of the system we are simulating and Ω is the set of all possible paths (failures and repairs) between time 0 and $\min(T_F, t)$. The unreliability can be written as [42]

$$\begin{aligned}
U(t) &= \int\limits_{\omega\in\Omega} I_\omega(T_F < t)df(\omega) \\
&= \int\limits_{\omega\in\Omega} I_\omega(T_F < t)\frac{df(\omega)}{dg(\omega)}dg(\omega) \\
&= \int\limits_{\omega\in\Omega} I_\omega(T_F < t)L(\omega)dg(\omega) \\
&= E_g\left[I(T_F < t)L\right],
\end{aligned}$$

where $I_\omega(\cdot)$ is the indicator function for the sample path ω and $L(\omega) = \frac{df(\omega)}{dg(\omega)}$ is the likelihood ratio. An unbiased estimate of $\hat{U}(t)$ is calculated using the new probability measure g by simulating N samples, $(I_1, L_1), (I_2, L_2), \ldots, (I_N, L_N)$ and computing

$$\hat{U}(t) = \frac{1}{N}\sum_{i=1}^{N} I_i L_i.$$

The confidence interval and relative error calculations are the same as standard simulation.

The next subsection describes an IS technique called *failure biasing* [42, 40], which we use to simulate rare events. While other IS techniques, (such as Splitting [17]) exist, we chose failure biasing for its simplicity and maturity.

5.2.1 Failure Biasing

To our knowledge, the most widely used form of IS is a technique called balanced failure biasing. While there exist separate techniques for balanced failure biasing under homogeneous [39] and non-homogeneous Poisson processes [42, 40, 41], we utilize two techniques that are applicable under both. The first uses a simple sampling method, called *uniformization* (or thinning [12]), to sample failure points from a non-homogeneous Poisson process. All points are sampled from a homogeneous Poisson process, which is "thinned" to match the target non-homogeneous process. The second method samples directly from the non-homogeneous Poisson process. As we will show, this method is only applicable under certain circumstances; we derive the sampling distribution for simulations with Weibull failure and repair distributions. As pointed out by Lewis and Shedler, the direct sampling approach is only applicable under well-behaved distributions [43]. The direct sampling approach requires a numerical solution to a non-linear equation. We have found that there are cases where standard numerical methods cannot properly solve the non-linear equation generated by directly sampling from the failure and repair distributions.

5.2.1.1 Uniformization-based Failure Biasing

The concept of uniformization was first developed by Lewis and Shedler in the late 1970's [43]. It was extended to simulating highly fault-tolerant systems in [40]. Suppose $\lambda(t)$ is the rate function (i.e. arrival rate) of a non-homogeneous Poisson process $\{N(t), t \geq 0\}$. If the available devices are indexed by AVAIL, then $\lambda(t) = \sum_{a \in \text{AVAIL}} h_a(c(D_a, t))$ is the system failure rate at time t, where h_a is the hazard function with respect to device a's failure distribution. Uniformization allows the simulation of a non-homogenous Poisson process with a homogeneous Poisson process, making the application of failure biasing much easier.

Consider the homogeneous Poisson process with rate β, $N_\beta(t)$. As long as $\beta \geq \lambda(t)$ for all valid values of t under simulation, this homogeneous process can be "thinned" to give us points in the non-homogeneous Poisson process with rate $\lambda(t)$. Event times in $N_\beta(t)$ are drawn from an exponential distribution with mean β. Suppose the i-th event time is drawn at time T_i. Then, with probability $\lambda(T_i)/\beta$, the point is accepted as an event in $N(t)$. Otherwise, the point is rejected. The rejected points are often called *pseudo events*.

We simulate a highly fault-tolerant system by drawing repair times from their original distributions and by determining the device failures as follows. Assume the current simulation time is T_n and the n-th event has just been processed. If the system is in a non-failed state, then sample the next device failure from

$$\{F_1(c(D_1, T_n)), F_2(c(D_2, T_n)), \ldots, F_n(c(D_n, T_n))\},$$

where $F_i(c(D_i, t))$ is the failure distribution of device i with age $c(D_i, t)$ at time t. If one or more devices are in a failed state, then we activate uniformization.

Let $r(t)$ denote the next device repair completion time on or after time t. We draw an event time, T_E from Exp(β). If $r(T_n) < T_E + T_n$, then $T_{n+1} = r(T_n)$ and the next event is a repair. If $r(T_n) \geq T_E + T_n$, then the next event is a device i failure with probability

$$\frac{h_i(c(D_{T_{n+1}}, t))}{\beta}$$

or a pseudo-event with probability $1 - \frac{\lambda(t)}{\beta}$.

To accelerate failures under uniformization, we implement balanced failure biasing as follows. Instead of accepting a point in the Poisson process as a failure with probability $\lambda(t)/\beta$, we now accept a point as a failure event with probability P_{fb}. If the point is accepted as a failure and there are A available devices, then a device is chosen to fail with probability $1/A$ (this is the "balanced" aspect of balanced failure biasing). Failure events are accelerated by choosing β to be a rate much closer to the average repair rate and a P_{fb} that is not too close to 0 or 1. While no general guidelines exist for choice of P_{fb}, some studies show that $0.5 \leq P_{fb} \leq 0.9$ leads to good results [42]. We choose the appropriate values for each parameter using empirical analysis (cf. Section 6.3). Since we have changed the probability dynamics of the system, all estimates of unreliability will be biased. We unbias the estimates using the likelihood ratio.

Let $N_F(t)$ be the number of accepted device failure events up to time t and $N_P(t)$ be the number of pseudo-events up to time t. In addition, let F_n be the time of the n-th accepted failure, $F(n)$ be the index of the device that failed and P_n be the time of the n-th pseudo-event. The likelihood ratio in a simulation with mission time t is [40]

$$\left[\prod_{n=1}^{N_F(t)} \frac{\lambda_{F(n)}(f_n)/\beta}{P_{fb}/A}\right] \left[\prod_{n=1}^{N_P(t)} \frac{1 - \lambda(P_n)/\beta}{1 - P_{fb}}\right].$$

The algorithm for processing failures in a non-degraded state is shown in Algorithm 4. As expected, the process is very similar to that of standard simulation, with two exceptions. First, the next failure event is drawn from a truncated distribution, $F_{i'}^{-1}(U, c(D_{i'}, t))$, with respect to the current age of each device. Second, instead of returning 1 at a data loss event, the current likelihood ratio (L) is returned. As we have stated, L unbiases our estimate.

Algorithm 4 Processing failure events in non-degraded mode when simulating under balanced failure biasing.

```
1: $(t_F, i_F) \leftarrow \{(t', i') : \min t' \text{ where } t' = F_{i'}^{-1}(U, c(D_{i'}, t)),\ \forall i' \in \textsc{avail}\}$
2: if $\min(t_F, t_R) > T$ then
3:     return $0$
4: end if
5: $\textsc{failed} \leftarrow \textsc{failed} \bigcup \{i_F\}$
6: $\textsc{avail} \leftarrow \textsc{avail} \setminus \{i_F\}$
7: $\textsc{sectors} \leftarrow \text{get\_failed\_sectors}(\textsc{avail})$
8: if $\textsc{dl}(\textsc{failed}, \textsc{sectors}) = 1$ then
9:     return $L$
10: end if
11: $\textsc{repairtimes} \leftarrow \textsc{repairtimes} \bigcup \left\{R_{i_F}^{-1}(U) + t_F, i_F\right\}$
```

Algorithm 5 illustrates processing failures and repairs in degraded mode. All event times are drawn from an exponential distribution with scale parameter β. If the next event time occurs after the next scheduled repair, then the repair is scheduled. Otherwise, a random variate is drawn from the uniform distribution on the unit interval ($U(0,1)$). We use the failure biasing probability to determine if the event is a failure or a pseudo-event. A pseudo-event simply updates the likelihood ratio and leaves the system state intact. Failure processing is similar to that of standard simulation with the exception of updating the likelihood ratio.

5.2.1.2 Direct Sampling Under Weibull Distributed Failures and Repairs

The most precise way to estimate the resulting unreliability of a non-homogeneous Poisson process under IS is sampling event times from the actual failure and repair distributions. While the direct route is more intuitive and analogous to standard discrete event simulation, we found that this type of sampling is highly dependent on the underlying distributions and in some cases may not be possible. While we found that this technique works quite well under certain conditions[1], it is primarily used for verification

[1]In fact, this is the primary IS method used when all distributions are exponential

Algorithm 5 Processing failures and repairs in degraded mode when simulating under balanced failure biasing with uniformization.

```
1: $t \leftarrow \text{exponential}(\beta) + t$
2: $(t_R, i_R) \leftarrow \{(t', i') \in \textsc{repairtimes} : t' \text{ is the min time over all entries}\}$
3: // Next event is repair
4: if $t_R < t$ then
5:     $\textsc{failed} \leftarrow \textsc{failed} \setminus \{i_R\}$
6:     $\textsc{avail} \leftarrow \textsc{avail} \bigcup \{i_R\}$
7:     if $|\textsc{failed}| == 0$ then
8:         $\textsc{state} \leftarrow \text{OK}$
9:     end if
10:    // Next event is pseudo-event
11: else if $U(0,1) < P_{fb}$ then
12:    $L \leftarrow L \times \frac{1-\lambda(t)/\beta}{1-P_{fb}}$
13:    // Next event is failure
14: else
15:    $i_F \leftarrow \text{rand}(\textsc{avail})$
16:    $L \leftarrow L \times \frac{\lambda_{i_F}(t)/\beta}{P_{fb}/|\text{AVAIL}|}$
17:    $\textsc{failed} \leftarrow \textsc{failed} \bigcup \{i_F\}$
18:    $\textsc{avail} \leftarrow \textsc{avail} \setminus \{i_F\}$
19:    $\textsc{sectors} \leftarrow \text{get\_failed\_sectors}(\textsc{avail})$
20:    if $\textsc{dl}(\textsc{failed}, \textsc{sectors}) = 1$ then
21:        return $L$
22:    end if
23:    $\textsc{repairtimes} \leftarrow \textsc{repairtimes} \bigcup \left\{ R_{i_F}^{-1}(U) + t_F, i_F \right\}$
24: end if
```

of the uniformization estimates.

As with uniformization, the failure rate at time t is

$$\lambda(t) = \sum_{a \in \text{AVAIL}} h_a(c(D_a, t))$$

, where AVAIL indexes the available devices. Similarly, the repair rate at time t is $\mu(t) = \sum_{f \in \text{FAILED}} h_a(r(D_f, t))$, where FAILED indexes the failed devices. The total rate at time t is $\lambda(t) + \mu(t)$. Let $\Lambda(t) = \int\limits_0^t (\lambda(x) + \mu(x))dx$ be the integrated rate function of the process. Under exponential and Weibull, this integral is very easy to calculate. We use $\Lambda(t)$ to generate the individual points between events. Assume $x_1, x_2, \ldots, x_i$ are the event times of our non-homogeneous Poisson process up time x_i. The next event time, $X_{i+1} - X_i$, is independent of $x_1, x_2, \ldots, x_{i-1}$ and has the following cumulative distribution function

$$F(x) = 1 - \exp\left(\Lambda(x_i) - \Lambda(x_i + x)\right).$$

As we have described in Section 5.1, we can readily draw from this distribution function by drawing a uniform random number, U, on the unit interval to compute $X_{i+1} - X_i = F^{-1}(U)$. While this method is much more direct than uniformization, there are a few drawbacks. First, since the individual event rates change between events, $F(\cdot)$ must be inverted every time we draw an event. Second, as Lewis and Shedler have pointed out, there may be cases where $X_{i+1} - X_i$ is not a proper random variable and may be infinite. We have found empirically that this method works very well under exponentially distributed failure and repair rates and certain 2-parameter Weibull distributions, but is, in general, quite inefficient.

The underlying algorithms for processing failure events are very similar to the algorithms used for failure biasing under uniformization. There are two exceptions: repair events are not directly scheduled and the likelihood ratio (and the event times) is based on the actual instantaneous event rate $\lambda(t) + \mu(t)$ instead of β.

5.3 High-Fidelity Reliability (HFR) Simulator

We have designed and built High-Fidelity Reliability (HFR) Simulator, a Monte Carlo reliability simulator, enabling "apples to apples" reliability comparisons of MDS codes, flat codes, Weaver codes, and simple product codes (SPC). The HFR Simulator is *high-fidelity* in that it accurately simulates the reliability of erasure codes with irregular fault tolerance that can tolerate two or more disk failures, with regard to both disk and sector failures. The key to high-fidelity simulation is effective *bookkeeping*: tracking which disk and sector failures have occurred, and efficiently determining if the failures constitute a data loss event. It is distinguished from the simulator that Elerath and Pecht recently described [14], by its ability to simulate the reliability of arbitrary, linear multi-disk fault tolerant erasure codes. The inclusion of sector failure and scrub rate models, and the accurate modeling of disk rebuilds, makes the HFR Simulator a more accurate tool than the recent Markov models proposed for erasure codes with irregular fault tolerance [25].

The HFR Simulator evaluates the reliability of a single array and calculates system reliability using the techniques presented in Section 5.1. In this section we make the details presented in Section 5.1 more concrete by describing simulation architecture, which includes simulation input and supporting structures.

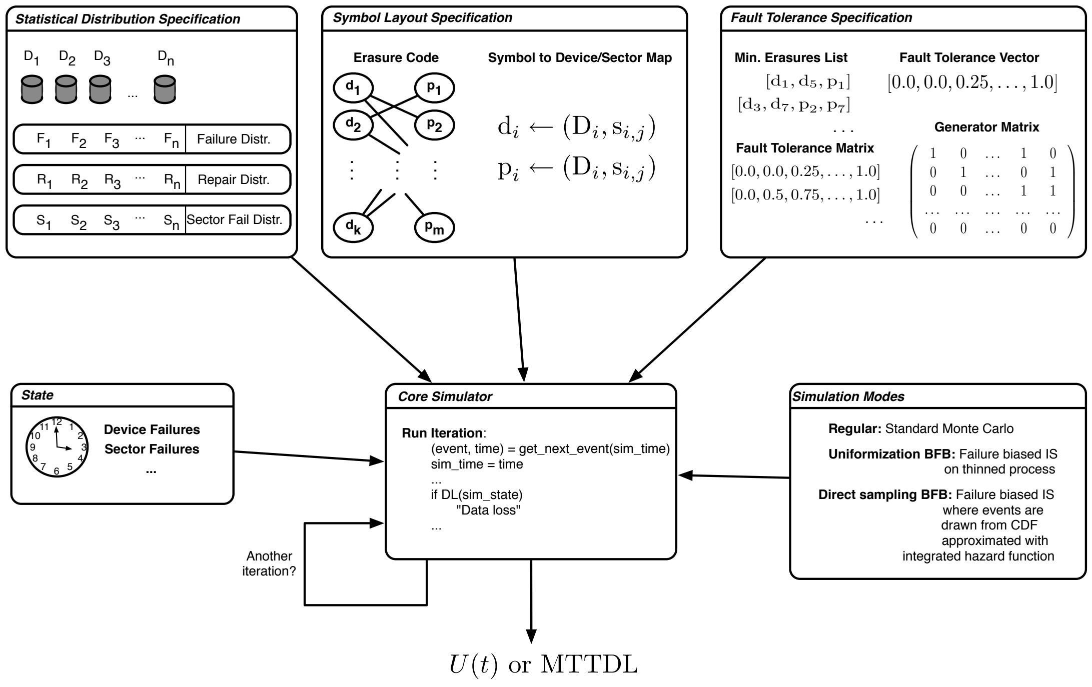
5.3.1 Architecture

A high-level architectural overview is shown in Figure 5.2. As shown, there are three major configurations used as input to the simulator:

Statistical Distribution Specification This configuration provides failure and repair information on the devices we are simulating. Each device may have its own unique failure and repair characteristics. In the current implementation, each device has a specific failure distribution, repair distribution and sector failure model.

Symbol Layout Specification In order to provide high-fidelity simulation, a mapping of individual code symbols to sectors is necessary. The symbol layout specification is a mapping from code symbols (data and parity) to strips. The individual sectors can easily be computed from this mapping.

Fault Tolerance Specification In previous reliability models, the underlying erasure code was assumed to be MDS, making the determination of data loss events trivial. Since our architecture accommodates any linear erasure code, more information is



Statistical Distribution Specification
D_1 D_2 D_3 ... D_n
F_1 F_2 F_3 ... F_n Failure Distr.
R_1 R_2 R_3 ... R_n Repair Distr.
S_1 S_2 S_3 ... S_n Sector Fail Distr.
Symbol Layout Specification
Erasure Code
Symbol to Device/Sector Map
d_1 d_2 d_k p_1 p_2 p_m
$d_i \leftarrow (D_i, s_{i,j})$
$p_i \leftarrow (D_i, s_{i,j})$
Fault Tolerance Specification
Min. Erasures List
$[d_1, d_5, p_1]$
$[d_3, d_7, p_2, p_7]$
Fault Tolerance Vector
$[0.0, 0.0, 0.25, \ldots, 1.0]$
Generator Matrix
Fault Tolerance Matrix
$[0.0, 0.0, 0.25, \ldots, 1.0]$
$[0.0, 0.5, 0.75, \ldots, 1.0]$
State
Device Failures
Sector Failures
Core Simulator
Run Iteration:
(event, time) = get_next_event(sim_time)
sim_time = time
if DL(sim_state)
"Data loss"
Another iteration?
Simulation Modes
Regular: Standard Monte Carlo
Uniformization BFB: Failure biased IS on thinned process
Direct sampling BFB: Failure biased IS where events are drawn from CDF approximated with integrated hazard function
$U(t)$ or MTTDL

Figure 5.2: Overview of the HFR Simulator.

needed to determine a data loss event. At a basic level, a code's MEL or generator matrix is sufficient to fully describe the fault tolerance of the code. We include the MEL, generator matrix and a new structure, called the *fault-tolerance matrix* for accurately and efficiently determining a data loss event.

While the statistical distribution specification and layout specifications are fairly straightforward, the fault tolerance specification deserves further attention. We detail how the structures are created in Section 5.3.2. In section 5.3.3 we show how these structures are used to create the data loss function, DL(), referenced in Section 5.1.

Once the simulator is initialized with a specific configuration file, a *simulation mode* is chosen. Currently, there are three modes of operation: regular, uniformization with balanced failure biasing and direct sampling with balanced failure biasing. Each of these techniques were described in Section 5.1. During the course of a single iteration, the simulator keeps track of current device failures, current sector failures and the life/rebuild clocks of each device. Depending on the mode of simulation, failure and repair schedules may also be indexed.

5.3.2 Fault Tolerance Matrix

Hafner and Rao constructed a vector of *conditional probabilities* in which the ith entry indicates the probability that i erasures are tolerated [25]. They used the conditional probabilities vector in a Markov model of the reliability of irregular erasure codes. They construct the conditional probabilities vector by performing matrix operations [24] on every combination of strip failures of size i up to the Hamming distance (or some other bound).

We construct the complement from the MEL. First, the MEL can be transformed into the *erasures list* (EL). The erasures list consists of every erasure pattern for a code. The EL is a super set of the MEL, and every element in it is either a minimal erasure or a super set of at least one minimal erasure. To transform a MEL into the corresponding EL requires only set and bitmap operations. These operations are computationally more efficient than using matrix methods to determine every erasure pattern as was done to calculate the conditional probabilities vector. Second, the *erasures vector* (EV) is to the EL what the MEV is to the MEL, and is easily determined given the EL. Finally, the *fault tolerance vector* is determined directly from the EV. The fault tolerance vector is the complement of the conditional probabilities vector; the ith entry indicates the probability that data is lost given i failures. The ith entry of the EV, e_i, is the number of erasure patterns of size i. For a code with n symbols, the ith entry of the fault tolerance vector, f_i, is thus $f_i = e_i/\binom{n}{i}$.

The fault tolerance vector is sufficient for describing the reliability of flat codes, but does not describe the reliability of irregular Vertical and HoVer codes. Irregular codes such as Weaver codes store multiple data symbols and parity symbols in each strip. Thus, sets of disk (strip) and sector (symbol) failures of some sizes may or may not lead to data loss; it depends on which disks and which sectors fail.

We have generalized the fault tolerance vector. The *fault tolerance matrix* describes the probability that a specific number of disk and sector failures leads to data loss. The j^{th} column of the i^{th} row of the fault tolerance matrix gives the conditional probability that j sector failures leads to data loss given i disks have failed. Indices for columns and rows begin at zero. Column zero of the fault tolerance matrix is effectively the fault tolerance vector (prepended with a zero). Row zero of the fault tolerance matrix is constructed in a similar manner to the fault tolerance vector, though the FTM is based on symbol failures rather than strip failures. Subsequent rows are somewhat more difficult to construct, because the EL must be reevaluated for every set of possible strip failures to determine how many combinations of additional symbol failures lead to data loss. This becomes quite intensive as codesize increases.

5.3.3 Bookkeeping

As we have described in Section 5.1, the DL(d, s) function takes a set of disk and sector failures as input and returns if a data loss event has occurred. There are three ways to determine if a set of failure events lead to data loss: perform a matrix rank test, compare the set of failures with the MEL, and precompute the fault tolerance matrix (FTM) to determine the probability that a certain number of disk and sector failures leads to data loss. In all three cases, separate tests may have to be performed for each distinct sector failure, since such failures are likely in distinct stripes. The HFR Simulator currently implements FTM, MEL and rank test processing.

5.3.3.1 Matrix rank bookkeeping

Disk (strip) and sector failures can be tested to determine if a data loss event has occurred using matrix methods (e.g., a rank test or some variant [24]). This test is effectively the same as attempting to rebuild failed data. A matrix rank test has to be performed for each set of concurrent failures.

A matrix rank test can be an expensive operation since it requires $O(k^2)$ operations. For small, flat codes matrix rank tests are not too expensive. However, HoVer codes such as the EVENODD parity-check array code has a very large k, because k is the number of data symbols, not data disks. For example, a 12 disk EVENODD code based on prime $p = 11$ has $k = 110$ data symbols and $m = 20$ parity symbols. A rank test for a matrix of this size is expensive.

5.3.3.2 MEL bookkeeping

The HFR Simulator can use the MEL of an erasure code to perform the bookkeeping necessary to determine if a combination of disk and sector failures leads to data loss. A bookkeeping copy of the MEL is maintained for each stripe containing an outstanding sector failure. This is necessary, since independent sector failures in different stripes may not affect one another. A bookkeeping copy is also maintained for all outstanding disk failures. Only one copy is needed because a single disk failure affects all stripes in a single array. We treat disk failures as correlated sector failures, thus we can determine if a data loss event has occurred by evaluating the intersection of a stripe's copy of the MEL against the bookkeeping MEL that exists for the disks. Fresh MEL copies are created and manipulated whenever there is a *period of interest*: a time period during which a sufficient number of failure events overlap for there to be risk of data loss.

As an example of how a bookkeeping copy of the MEL is updated by the simulator, consider the MEL for the (5,3)-FLAT listed in Section 3.4. Each time a failure occurs, a bookkeeping copy of the MEL is updated by removing the failed symbol from each minimal erasure in which it is a member. An empty minimal erasure in the bookkeeping copy means that data is lost. Consider the the bookkeeping MEL if symbol s_7 fails and then symbol s_4 fails: after symbol s_7 fails, the bookkeeping MEL is $\{(s_4)$, (s_0, s_1, s_4), (s_0, s_1), (s_0, s_2, s_6), (s_0, s_3, s_5), (s_1, s_2, s_3), (s_1, s_5, s_6), (s_2, s_4, s_5), (s_2, s_5), (s_3, s_4, s_6), $(s_3, s_6)\}$; and, after symbol s_4 fails, the bookkeeping MEL is $\{()$, (s_0, s_1), (s_0, s_1), (s_0, s_2, s_6), (s_0, s_3, s_5), (s_1, s_2, s_3), (s_1, s_5, s_6), (s_2, s_5), (s_2, s_5), (s_3, s_6), $(s_3, s_6)\}$. The () entry in the final bookkeeping MEL indicates that data loss has occurred.

As with Tanner graphs, minimal erasures are represented as bitmaps (i.e., as an integer less than 2^{k+m}). Using this data structure, deleting an element from a set involves bitwise-XOR, adding an element to a set is bitwise-OR and testing if an element is in a set uses bitwise-AND. Each bookkeeping MEL is stored as a dictionary keyed on minimal erasures. Each entry in a bookkeeping MEL is initialized to be the minimal erasure (i.e., the initial value is the key). Another dictionary, the bookkeeping index, keyed on code symbols, lists all minimal erasures for which the symbol is a member. When some symbol is erased or recovered, the bookkeeping index is used to determine which entries in the pertinent bookkeeping MEL to update. If a value in a bookkeeping MEL becomes zero, then a data loss event has occurred.

It is possible to reduce the size of the MEL used for bookkeeping. The reliability is dominated by the minimal erasures with lowest weight. For flat codes, a weight cutoff distance above which minimal erasures are unnecessary for bookkeeping works. A cutoff of two greater than the Hamming distance works for the parameters we have used. For HoVer codes, care must be taken since every disk (strip) failure results in multiple sector (symbol) failures. The Hamming distance needs to be multiplied by the number of elements per strip for such codes. In general, this does not prune much of the MEL. Instead, processing of the MEL similar to that which is necessary to determine the fault tolerance matrix (cf. Section 5.3.3.3) can produce a pruned MEL that is sufficiently accurate.

5.3.3.3 FTM bookkeeping

The HFR Simulator can perform bookkeeping using the fault tolerance matrix (FTM). This requires additional analysis of the code *a priori* to determine the FTM. The *fault tolerance matrix* is used to determine if sector failures in conjunction with strip failures leads to data loss.

Consider a traditional double disk fault tolerant parity check array code such as EVENODD. First, consider the fault tolerance vector that only accounts for disk failures. For any EVENODD code, the FTV is $(0, 0, 1)$ because the code tolerates all one and two disk failures, but no three disk failures.

Now, consider the fault tolerance matrix for an EVENODD code. The first row of the FTM begins $(0, 0, 0, x, \dots)$, indicating that with probability x, a triple symbol failure leads to data loss. The second row begins $(0, 0, y, \dots)$, indicating that with probability y, a two sector failures in conjunction with a single strip failure leads to data loss. The third row begins $(0, 1, 1, \dots)$, indicating that with probability 1, any additional sector failure in conjunction with a double strip failure leads to data loss. Finally, the fourth row begins $(1, 1, \dots)$ and indicates that the loss of any three strips leads to data loss.

Similar to the prefix of the MEL, only the first couple non-zero entries in each row of the FTM are required for accuracy. Truncating the fault tolerance vector and matrix reduces the amount of preprocessing necessary before running a simulation.

The layout of an erasure code affects the construction of the FTV and the FTM. If a code has a static layout, i.e., a fixed assignment of code symbols to disks, then the construction methods described above are correct. However, if a code has a rotated layout, i.e., the assignment of code symbols to disks is staggered from stripe to stripe to ensure that load is balanced across all devices, and the FTV and FTM need to be modified. Specifically, the likelihood that disk failures, exclusive of sector failures, leads to data loss is modified. First, consider the FTV for a code with $n = 10$ symbols and an MEV of $(0, 1, \dots)$, i.e., with one minimal erasure of size two and so on. With a static placement, the FTV is $(0, \frac{1}{45}, \dots)$. With a rotated placement, the fact that each stripe is rotated leads to the minimal erasure repeating on 10 pairs of devices, and so the FTV is $(0, \frac{10}{45}, \dots)$. Similar reasoning is applied to correctly construct the FTM. Note that a rotated placement of an irregular erasure code reduces its reliability. In some sense, the code becomes "less" irregular because of the rotational symmetry across stripes.

The HFR Simulator uses the probabilities in the FTV and FTM to determine the likelihood that a given set of disk and sector failures leads to data loss. The FTV is used for regular codes and flat irregular codes, and the FTM is used for HoVer and Vertical irregular codes (i.e. array codes).

5.4 Implementation

Currently, there are two versions of the HFR Simulator: HFRS v.1 and HFRS v.2. The underlying framework of HFRS v.1 and HFRS v.2 are extremely similar. Both versions implement the novel bookkeeping structures, have the ability to model heterogeneous devices with varied failure and repair distributions, account for sector errors and have the ability to keep track of the critically exposed region during rebuild. The primary difference between the versions lies in the reliability metric and efficiency. While HFRS v.1 is relatively inefficient and reports MTTDL (or DLE per PB-YEAR), HFRS v.2 is very efficient and reports the probability of data loss for a specified mission time.

Both HFRS v.1 and HFRS v.2 are written in Python and are each around 2500 lines of code (not including supporting modules such as the MEL and bit-matrix operations). HFRS v.1 relies on Python's `random` library for all random number generation and statistical distributions. HFRS v.2 uses the open source multi-precision math library (`mpmath`) for random number generation and floating point calculations. Due to the lack of flexibility in most statistical packages for Python, I created a statistical distribution module that implements the Weibull distribution specifically for HFRS v.2.

5.4.1 HFRS v.1

The original implementation of the HFR Simulator, called HFRS v.1, was created in 2007 and relies on unbound discrete-event simulation. The basic simulation loop resembles Algorithm 3. Instead of drawing sector failures during a critical mode, sector failures are drawn individually, similar to Elerath and Pecht's simulator [14]. HFRS v.1 has the ability to report reliability in terms of MTTDL and data loss events per petabyte year DLE per PB-YEAR. A sensitivity analysis was performed using codes with a Hamming distance of at most 3.

The fundamental limitation of HFRS v.1 was efficiency. HFRS v.1 had two major inefficiencies. First, as stated in Section 5.1, standard simulation of highly fault-tolerant codes is extremely inefficient. This limitation is overcome by the creation of HFRS v.2, which incorporates importance sampling techniques to efficiently simulate highly fault tolerant systems. Where HFRS v.1 only has the ability to simulate 2 disk-fault tolerant codes, HFRS v.2 can efficiently simulate up to 4 disk fault tolerant codes. Second, treating both disk failures and sector failures as first class events can be quite inefficient, since a great deal of sector failures are drawn between disk failures. Most of the sector failure events do not make a significant contribution to reliability. As we have argued, the current disk and sector failure models are most efficiently modeled by drawing sector failures in the critical mode. In the event that multiple, concurrent sector failures is a dominant failure mode, simulations must use the sector failure mechanism of HFRS v.1.

5.4.2 HFRS v.2

HFRS v.2 was created to overcome the inefficiencies of HFRS v.1. There are three major differences between the implementations: the outer simulation loop, ability to efficiently simulate rare events and the use of a different, but comparable sector failure framework. First, HFRS v.2 is a mission time-based simulator; thus, all reliability estimates are reported in probability of data loss over a time horizon. Second, under a mission time-based simulation, importance sampling techniques are used to efficiently simulate highly fault-tolerant codes. Both HFRS v.1 and HFRS v.2 have comparable performance for single disk fault tolerant systems; HFRS v.2 tends to be more efficient for double disk fault tolerant and beyond. Finally, to further increase efficiency and maintain balanced failure rates, sector failures are treated as second-class events in HFRS v.2 and the scrubbing interval is incorporated into the sector failure probability. That is, sector failures are only drawn directly after a first-class event (i.e. disk failure) and the sector failure probability distribution incorporates both latent sector errors and the corresponding scrubbing interval. Note, that this sector failure model can effectively perform the same analysis as HFRS v.1 by drawing for sector failures after each disk failure event. In the current implementation, sector failures are only drawn during the critical mode.

5.4.3 Supporting Modules

I created a library to implement analytical calculations, called `markov_model`. The library has the ability to simulate and analytically solve arbitrary Markov models. I have implemented a special purpose modeling language that allows the evaluation of arbitrary models. Additionally, the library implements balanced failure biasing in a Markov chain. The library solves steady state measures (i.e. MTTDL) using the linear algebra package in NumPy and transient measures (i.e. probability of data loss) using the Runge-Kutta numerical methods implemented in SciPy. We use `markov_model` to verify the accuracy of the simulator.

5.4.4 Use of HFRS v.1 and HFRS v.2

The techniques used to create HFRS v.1 and HFRS v.2 have been used in a variety of projects outside of this work. The sensitivity analysis performed using HFRS v.1 highlighted the power of simple product codes in the face of sector errors, which led to the adoption of such codes in the Pergamum system [58]. The reliability calculations used to verify the encoding strategy in Pergamum used an updated codepath of HFRS v.1. In addition, recent work in large-stripe erasure codes also relies on HFRS v.1 [62]. The study performed in Chapter 7 relied heavily on HFRS v.1 [20]. The work in power-aware coding (cf. Chapter 8) relies on HFRS v.2.

Some of the supporting modules have been used outside of the HFR Simulator work. The `markov_model` package was used to estimate reliability in two other UCSC projects [6, 19].

5.5 Summary

In this chapter we have described methods used to perform standard simulation of erasure-coded storage systems. We have combined simulation, importance sampling techniques and novel bookkeeping structures into a single simulation framework for evaluating the reliability of erasure-coded storage systems: the High-Fidelity Reliability (HFR) Simulator. We have described the implementation of two versions of the HFR Simulator. Both versions represent the most advanced simulation environments for erasure-coded storage systems. The focus of the next chapter is a detailed, empirical study based on HFRS v.1 and HFRS v.2.

Chapter 6

Evaluation of the High-Fidelity Reliability (HFR) Simulator

In this chapter, we evaluate the HFR Simulator. The evaluation is split into seven parts that rely on both versions of the HFR Simulator: HFRS v.2 and HFRS v.1. While, HFRS v.2 is much more efficient that HFRS v.1, a sensitivity analysis was performed using HFRS v.1, which is interesting and worth mentioning.

We begin by describing the parameters used in all of the simulations. Parameters include device failure distributions, device repair distributions, sector error distributions and erasure codes. This evaluation is mostly exploratory and consists of a detailed validation, sensitivity analysis and reliability comparison of 17 distinct erasure codes. The seven parts of this evaluation are as follows:

Validation of HFRS v.1 and HFRS v.2. We validate the HFRS v.1 simulator against the Elerath and Pecht's simulator [14] and the Markov model shown in Figure 4.5. Since HFRS v.2 is more sophisticated than any previous reliability simulator, we must validate simple HFRS v.2 configurations against comparable Markov models.

Sensitivity of IS parameters. As mentioned in Chapter 5, the parameters used for importance sampling must be found empirically. We explore a spectrum of IS parameters under varied fault tolerance and constant failure/repair distributions to determine both parameter sensitivity and appropriate parameters for simulation.

Sensitivity of failure and repair distributions. One of the main arguments against the use of Markov models is that exponential distributions do not fully capture actual failure and repair distributions. Current data suggests that disk failure and repair can be modeled by Weibull distributions. We compare the reliabilities of Weibull distributed and exponentially distributed failures and repairs. The comparison is as follows. We choose a Weibull failure or repair distribution with mean λ. We compare the reliability of this distribution under simulation with an exponential distribution with mean λ. This allows us to determine if time dependence has an effect on reliability.

Code sensitivity under varied rebuild rate, scrub rate and sector failure rate. We compare the relative reliability of four structurally different codes as a function of rebuild rate, scrub rate and sector failure rate. This comparison allows us to determine sensitivity of code structure to specific failure and repair characteristics. We also study the reliability of SPC codes, which if used correctly, masksector errors.

Comparison of sector failure models. The relative accuracy of the sector error models discussed in Section 4.3 are compared. We compare the reliability approximations given by Markov models to that of simulation. Such a comparison allows us to determine the difference between a model that includes the critical region of the rebuild process and a model that ignores the critical region.

Sensitivity of bookkeeping structures. The HFR Simulator has novel structures for identifying of data loss events. These bookkeeping structures, described in Section 5.3.3, result in different running times. We compare the resulting reliability calculation of each structure to determine the relative accuracy of each technique.

Apples-to-apples reliability comparison. Finally, we perform the first apples-to-apples, high-fidelity comparison of a variety of erasure codes. The results are apples-to-apples because each code is evaluated under the same framework, opposed to traditional modeling, which may require a different model for each system instance. In the analysis, we compare the relative reliabilities of MDS codes, flat XOR-based codes and XOR-based array codes.

distr.	**scale**	**shape**	**location**
Disk Failure	$461,386$	1.12	0.0
Disk Repair	12.0	2.0	6.0

Table 6.1: Parameters for the Weibull distributions used to model disk failures and repairs.

All measurements are taken from HFRS v.1, HFRS v.2 and our `markov_model` module. The HFRS v.1 measurements represent a study that was conducted roughly two years prior to the development of HFRS v.2. The HFRS v.1 numbers represent MTTDL and are presented to validate the design of the simulator and present a few key results found using HFRS v.1. Most of the analysis is carried out using HFRS v.2, which is much more efficient than HFRS v.1. HFRS v.2 reports reliability as the probability of data loss within a specified mission time (unreliability).

6.1 Simulation Parameters

We use the same disk failure and repair parameters for simulations with the HFR Simulator as Elerath and Pecht (cf. Table 2 in [14]). The disk failure distribution is based on failure data and is modeled as a Weibull with a scale of $461,386$, a shape of 1.12 and location 0. The disk repair distribution is also Weibull with scale 12, shape 2.0 and location 6. Elerath and Pecht use this repair distribution to enforce a 6 hour minimum on rebuilds. The high shape parameter ensures that rebuilds will not take too long. We rely on the scale parameters of these distributions where exponential distributions are used.

A reasonable failure distribution for sector failures has yet to surface. The current state-of-the-art derives sector failure rates from the BER and a given workload. All current models assume the rates are exponentially distributed [29, 56, 14].

HFRS v.1 uses the same sector error and scrub model as Elerath and Pecht [14]. Latent sector errors are exponentially distributed with scale parameter 9259 and scrubs are Weibull distributed with location 6, scale 168 and shape 3.

For HFRS v.2, we use the model presented by Iliadis *et al.* for three reasons.

First, the effect of scrubbing on sector failures is included into a single probability, P_S (cf. Section 4.3.2), which makes the model easy to incorporate into simulations that use failure biasing. Second, the model is extensible: we have shown in Section 4.3.2 how time dependence could be added into the model. Finally, it enforces a more strict bound on the scrubbing interval than the model developed by Elerath and Pecht. Elerath and Pecht attempt to enforce bounds on the scrubbing interval by adjusting the shape and location of a Weibull distribution.

The parameters of the latent sector error and scrub model for HFRS v.2 are as follows. We base our sector failure model on the probability proposed by Iliadis *et al.*:

$$P_S(t) = \left(1 - e^{-h(t \bmod T_S)}\right) P_e.$$

Instead of performing calculations in years, our calculations are in hours. We use this model with a load (h) of 0.0047, a scrub interval (T_S) of 168 hours and a sector error probability (P_e) of 4.096×10^{-11}. These parameters are consistent with the parameters used in [14].

The parameters used for balanced failure biasing in HFRS v.2 are as follows. The β parameter used for sampling from an exponential distribution under uniformization is chosen as $\max\{h(t)\} \cdot 2$, $0 \leq t \leq 3/\mu$, where $h(\cdot)$ is the hazard function with scale μ (μ is the scale of the repair distribution) and $3/\mu$ is an upper bound on the repair time. This particular parameter was chosen experientially, since it appears to give good estimates under the systems we analyze. Similar parameters were used in other studies that use uniformization [40]. Our failure biasing parameter range of 0.2 to 0.5 are also in accordance with previous studies. We found that under both direct sampling and uniformization this range yields good estimates.

We model 300 GB disks with 512 B sectors, i.e., there are $s = 585937500$ sectors per disk. We use a concurrent rebuild policy for all multi-disk fault tolerant codes. The HFR Simulator uses the FTV and/or FTM for bookkeeping in most experiments. The experiments which use the MEL for bookkeeping list this explicitly.

6.1.1 Erasure Codes used in the Analysis

The fault tolerance, rate, and reliability are listed for each code under evaluation. Fault tolerance is measured in terms of the Hamming distance, MEV, FTV, and/or

code	Hamming distance	rate	number of disks
(16,4)-MDS	5	0.80	20
(17,3)-MDS	4	0.85	20
(15,5)-FLAT	3	0.75	20
(16,4)-FLAT	2	0.80	20
(5,3)-MDS	4	0.63	8
(6,2)-MDS	3	0.75	8
(7,1)-MDS	2	0.88	8
(36,12)-EVENODD	3	0.75	8
(36,12)-RDP	3	0.75	8
(42,12)-SPC	4/2	0.75	8
(4,4)-FLAT	4	0.50	8
(5,3)-FLAT	2	0.63	8
(6,2)-FLAT	2	0.75	8
(8,8,1)-WEAVER	2	0.50	8
(8,8,2)-WEAVER	3	0.50	8
(8,8,3)-WEAVER	4	0.50	8
(35,14)-XCODE	3	0.71	7

Table 6.2: List of evaluated erasure codes.

FTM. Hamming distance d is a coarse-grained fault tolerance measure for irregular codes and the primary measure for MDS codes. We sometimes list two numbers for the Hamming distance, e.g., a/b. The first number a corresponds to the traditional Hamming distance with regard to symbol failures, and the second number, b, corresponds to the "disk" Hamming distance with regard to strip/disk failures. For some Vertical and HoVer codes, the value of b can be less than that of a. Rate characterizes the space efficiency of the code; it is the fraction of symbols in the code that are data symbols (rate cannot exceed $k/(k+m)$).

Table 6.2 lists the codes evaluated in the reliability comparison. The Hamming distance, rate and number of disks are given for each code.

Five MDS codes are used in the evaluation: (16,4)-MDS, (17,3)-MDS, (5,3)-MDS, (6,2)-MDS and (7,1)-MDS. These are flat codes, i.e., each strip consists of a single symbol. (7,1)-MDS is the same as an 8 disk RAID 4 code.

code	fault tolerance matrix (ftm)
(36,12)-EVENODD	0.0000, 0.0000, 0.0000, 0.0018, ...
	0.0000, 0.0000, 0.0469, 0.1344, ...
	0.0000, 1.0000, 1.0000, 1.0000, ...
	1.0000, 1.0000, 1.0000, 1.0000, ...
(36,12)-RDP	0.0000, 0.0000, 0.0000, 0.0021, ...
	0.0000, 0.0000, 0.0440, 0.1500, ...
	0.0000, 1.0000, 1.0000, 1.0000, ...
	1.0000, 1.0000, 1.0000, 1.0000, ...
(35,14)-XCODE	0.0000, 0.0000, 0.0000, 0.0019, ...
	0.0000, 0.0000, 0.0383, 0.1314, ...
	0.0000, 1.0000, 1.0000, 1.0000, ...
	1.0000, 1.0000, 1.0000, 1.0000, ...
(42,12)-SPC	0.0000, 0.0000, 0.0000, 0.0000, 0.0016, ...
	0.0000, 0.0000, 0.1250, 0.3484, 0.6034, ...
	1.0000, 1.0000, 1.0000, 1.0000, 1.0000, ...

Table 6.3: Fault tolerance matrix for parity-check array codes. Each row corresponds to a disk failure starting from 0 and each column corresponds to a correlated sector failures starting from 0.

Four parity-check array codes are used in the evaluation: (36,12)-EVENODD, (36,12)-RDP, (42,12)-SPC, and (48,16)-XCODE. The EVENODD and RDP code are both constructed for prime number 7. The EVENODD code is shortened by one column so as to only use 8 disks rather than 9. The X-code is constructed with 7 disks, each of which has 5 data symbols and 2 parity symbols per strip. Note, that the X-code has a slightly lower rate than the other parity-check array codes. We used $n = 7$ disks for the X-code to make its rate as similar to the other parity-check array codes as possible. The (42,14)-SPC has 6 symbols in each strip; the first 7 disks each have 5 data symbols and one parity symbol, and the last disk has 6 parity symbols. This SPC construction was chosen so that it has the same rate as the EVENODD and RDP codes.

The Hamming distance of the SPC code at the symbol level is 4. However because of the parity symbol at the end of each data strip, the "disk Hamming distance" is only 2. The FTM for each parity-check array code is given in Table 6.3.

A few Weaver codes are included in the comparison [25]. These codes are

code	fault tolerance matrix (ftm)
(8,8,1)-WEAVER	0.0000, 0.0000, 0.0667, 0.2000, . . .
	0.0000, 0.1428, 0.3406, 0.5604, . . .
	0.2857, 0.5238, 0.7402, 0.8961, . . .
	0.7142, 0.8857, 0.9746, 1.0000, . . .
	. . .
(8,8,2)-WEAVER	0.0000, 0.0000, 0.0000, 0.0143, . . .
	0.0000, 0.0000, 0.0440, 0.1483, . . .
	0.0000, 0.1190, 0.3073, 0.5584, . . .
	0.2857, 0.5428, 0.8158, 1.0000, . . .
	. . .
(8,8,3)-WEAVER	0.0000, 0.0000, 0.0000, 0.0000, 0.0044, . . .
	0.0000, 0.0000, 0.0000, 0.0110, 0.0579, . . .
	0.0000, 0.0000, 0.0346, 0.1662, 0.4949, . . .
	0.0000, 0.1143, 0.4413, 0.9833, 1.000, . . .
	. . .

Table 6.4: Fault tolerance matrix for Weaver codes. Each row corresponds to a disk failure starting from 0 and each column corresponds to a correlated sector failures starting from 0.

labeled as (k,k,t)-WEAVER codes. They consist of k data symbols, $m = k$ parity symbols, and are based on the Weaver construction for t of 1, 2 and 3, which represents the disk fault-tolerance of each code [25]. Each of the Weaver codes has a rate of 0.5. The FTM for each Weaver code evaluated in the comparison is given in Table 6.4.

Five irregular XOR-based erasure codes are used in the evaluation: (4,4)-FLAT, (5,3)-FLAT, (6,2)-FLAT, (15,5)-FLAT and (16,4)-FLAT. The RAID 10 code is often referred to as “replicated striping”. The flat codes were found via prior work by Wylie and Swaminathan [65]. Table 6.12 lists the parity bitmaps (cf. Section 3.4.2) and FTV of these codes. In Table 6.2 the Hamming distance listed corresponds to the first non-zero entry in the FTV. All of the flat codes are static (i.e., the codes are not rotated to balance load), though the HFR Simulator has the ability to handle such layouts.

code	parity bitmaps	fault tolerance vector (ftv)
(4,4)-FLAT	$7, 11, 13, 14$	$(0.000, 0.000, 0.000, 0.200, 1.000)$
(15,5)-FLAT	$255, 3855, 13107, 23756, 25941$	$(0.000, 0.000, 0.028, 0.151, 0.479, 1.000)$
(16,4)-FLAT	$511, 7711, 26215, 43691$	$(0.000, 0.026, 0.149, 0.479, 1.000)$
(5,3)-FLAT	$7, 11, 29$	$(0.000, 0.036, 0.286, 1.000)$
(6,2)-FLAT	$15, 51$	$(0.000, 0.250, 1.000)$

Table 6.5: Fault tolerance vector for FLAT codes.

6.2 Validating Simulation via Markov Models

Here we validate both versions of the HFR Simulator using comparable Markov models. In Chapter 4 we argued that Markov models are not appropriate for accurate, high-fidelity reliability analysis. Our validation tests the main paths of the HFR Simulator by simulating simple systems: systems with exponential failure and repair distributions, no critical mode sector errors and MDS codes. The validation provides confidence in the results presented in this section.

6.2.1 Validation of HFRS v.1

To validate the correctness of the implementation of the HFRS v.1, we reproduce results from another simulator and reproduce results of some Markov models. Elerath and Pecht's results in Table 3 of [14] were reproduced with HFRS v.1, which are for a single disk fault tolerant MDS code, includes latent latent sector failures and scrubbing, and are the result of Monte Carlo simulation. This is significant validation because the Markov model validation described below only validates the correctness of the simulator using exponential distributions for limited scenarios.

We validated the single-disk fault tolerant case without sector failures and scrubbing with a traditional Markov model (cf. Figure 4.1 in Chapter 4). The HFR Simulator produces numbers that are $\approx 10\%$ less than the analytic results due to the Markov model. We used the following parameters for the Markov model : $n = 8$, $\lambda = 1/461386$ (disk failure rate), $\mu = 1/12$ (disk repair rate). The simulator produced an MTTDL of $2.95 \times 10^8 \pm 1.83 \times 10^7$, while the Markov model produced an MTTDL of

3.17×10^8.

The single-disk fault tolerant case with sector failures was validated against a Markov model. Figure 4.5 in Chapter 4 illustrates the Markov model. The Markov model was validated with the following parameters: $n = 8$ (number of disks), $\lambda_d = 1/461386$ (disk failure rate), $\mu_d = 1/12$ (disk rebuild ratE), $\lambda_s = 1/9259$ (sector failure rate, per disk), $\mu_s = 1/168$ (failed sector scrub rate), and $s = 585937500$ (number of sectors per disk). The parameter ρ distributes the likelihood of a sector failure occurring during rebuild between the data loss state and the latent sector failed state. The simulator produced an MTTDL of $4.944 \times 10^5 \pm 5.54 \times 10^4$, while the Markov models produced an MTTDL of 5.201×10^5.

6.2.2 Validation of HFRS v.2

In order to validate the accuracy of the HFRS v.2, the reliability of five systems are evaluated under simulation and analytically. In order to get accurate measurements from the analytical models, systems with MDS codes are evaluated and do not account for the critical region in the rebuild process. Throughout this subsection, we assume that all disk failures and repairs are exponentially distributed.

Two types of models are evaluated in this section. First, the reliability under a process where the state transitions are disk rebuilds and repairs is considered (cf. Figure 4.2-(c) in Chapter 4). Second, sector failures are incorporated into the process (cf. Figure 4.4 in Chapter 4). The resulting reliabilities of standard simulation (SIMREG), balanced failure biasing under uniformization (BFBUNIF), balanced failure biasing with direct sampling of the processes inverted rate function (BFBINV) and solving via the Kolmogorov equations are compared. These are the techniques covered in Chapter 5.

The results presented in this section serve three purposes. First, the modes of simulation are validated by comparing to a comparable analytic model. Second, the relative accuracy of each simulation technique is compared. Finally, the performance boost gained by simulating under importance sampling is illustrated.

code	**sim. type**	**failure prob.**	**rel. err. (%)**	**run time (s)**	**analytical prob.**
	BFBUNIF	7.06×10^{-15}	49.18	3.08×10^{2}	
(16,4)-MDS	BFBINV	6.87×10^{-15}	12.18	8.00×10^{2}	6.49×10^{-15}
	SIMREG	—	—	—	
	BFBUNIF	6.69×10^{-11}	12.28	2.85×10^{2}	
(17,3)-MDS	BFBINV	6.40×10^{-11}	5.51	7.68×10^{2}	6.46×10^{-11}
	SIMREG	—	—	—	
	BFBUNIF	8.96×10^{-13}	11.35	5.88×10^{1}	
(5,3)-MDS	BFBINV	9.43×10^{-13}	4.19	1.70×10^{2}	9.35×10^{-13}
	SIMREG	—	—	—	
	BFBUNIF	2.10×10^{-8}	4.33	5.19×10^{1}	
(6,2)-MDS	BFBINV	2.17×10^{-8}	2.23	1.36×10^{2}	2.17×10^{-8}
	SIMREG	1.00×10^{-7}	164.50	2.25×10^{3}	
	BFBUNIF	2.83×10^{-4}	7.28	—	
(7,1)-MDS	BFBINV	2.76×10^{-4}	1.06	7.89×10^{1}	2.76×10^{-4}
	SIMREG	2.58×10^{-4}	14.48	2.26×10^{3}	

Table 6.6: Simulator validation against an analytical model without sector failures (cf. Figure 4.2-(c) in Chapter 4).

Table 6.6 provides a comparison of the reliability and performance of the simulator for 5 MDS codes. We run the failure biasing simulations for $100K$ iterations and the standard simulations for $10M$ iterations. If the relative error exceeds 20%, then the simulation is re-run, otherwise, we report the estimated reliability. For validation, the simulated estimates are compared to the corresponding analytical estimates. Additionally, we provide the running times of any simulation that yields a non-zero estimate. It is obvious from the running times that the importance sampling techniques provide a great deal of efficiency compared to standard simulation. As an example, the HFR Simulator can simulate a 2 disk fault tolerant system and obtain a much more accurate number in 2% of the time required under standard simulation. Additionally, we can get reasonable estimates for systems that cannot be simulated via standard simulation in a matter of minutes.

As shown in Table 6.6 the fast simulation techniques leads to quite accurate reliability estimates, even when the underlying system is highly fault tolerant. Additionally, these estimates are obtainable in a reasonable amount of time. Obviously, BFBINV gives the most accurate results for 3 and 4 disk fault-tolerant systems. This is because the waiting times are sampled directly from an exponential distribution. The higher variance and slightly less accurate measurements seen in the BFBUNIF simulations are due to the fact that the process first samples from a thinned homogeneous process instead of direct sampling. In the case of 1, 2 and 3 disk fault-tolerant systems, the thinned process typically returns estimates with a low relative error. In the case of a 4 disk fault-tolerant system, many trials may be required to get a reasonable estimate.

The high relative error is most likely due to the excessive number of transitions in a single iteration of a 4 disk fault tolerant system. This limitation is quite intuitive. At every failure transition, the likelihood ratio is updated. As the fault tolerance increases, the path lengths to the failure state increase (i.e. more failure and repair transitions). These excessive state transitions may negatively impact the likelihood ratio, leading to an estimate with high variance. This limitation is well known and methods for dealing with it in Markovian processes generally "cancel" terms in the likelihood ratio. We do not know of any methods for dealing with this issue in non-Markovian processes; thus, we re-run must the 4 disk fault tolerant reliability simulations until we get a reasonable estimates.

We believe that a 4 disk fault tolerant system may be the limit for the failure biasing techniques and more advanced techniques will be required to accurately estimate the reliability of systems with higher fault tolerance. One option is to augment the failure biasing techniques with multi-level splitting techniques.

Table 6.7 contains the same simulations as Table 6.6 with latent sector failures. As with the data shown in Table 6.6 the estimates are quite close to the analytical estimates. We find an interesting performance consequence of including sector failures in standard simulation. The running time of 10M iterations of the (7,1)-MDS code takes more than twice the amount of time as the simulation that does not include sector failures. This is due to the additional bookkeeping required when checking for sector errors. When there are one or more concurrent disk failures, all other disks participating

code	**sim. type**	**failure prob.**	**rel. err. (%)**	**run time (s)**	**analytical prob.**
	BFBUNIF	6.48×10^{-12}	23.43	3.10×10^{2}	
(16,4)-MDS	BFBINV	7.20×10^{-12}	16.08	8.00×10^{2}	7.00×10^{-12}
	SIMREG	—	—	—	
	BFBUNIF	4.44×10^{-8}	10.24	2.99×10^{2}	
(17,3)-MDS	BFBINV	4.99×10^{-8}	7.06	7.65×10^{2}	5.03×10^{-8}
	SIMREG	—	—	—	
	BFBUNIF	5.85×10^{-10}	19.53	6.20×10^{1}	
(5,3)-MDS	BFBINV	7.42×10^{-10}	12.61	1.69×10^{2}	7.59×10^{-10}
	SIMREG	—	—	—	
	BFBUNIF	1.23×10^{-5}	10.67	5.99×10^{2}	
(6,2)-MDS	BFBINV	1.14×10^{-5}	6.42	1.41×10^{2}	1.16×10^{-5}
	SIMREG	1.19×10^{-5}	15.08	1866.05	
	BFBUNIF	7.23×10^{-2}	4.65	5.41×10^{1}	
(7,1)-MDS	BFBINV	7.19×10^{-2}	3.14	9.88×10^{1}	7.18×10^{-2}
	SIMREG	7.18×10^{-2}	0.19	5804.12	

Table 6.7: Simulator validation against an analytical model with sector failures (cf. Figure 4.4 in Chapter 4).

in the rebuild operation are checked for a sector error. Since the primary failure mode of (7,1)-MDS is a single disk failure with a latent sector failure, these sector failure checks dominate the running time of each iteration.

This validation gives confidence that the HFR Simulator reports accurate reliability estimates for systems having at most 4 disk fault tolerance. Beyond 4-disk fault tolerance, additional techniques, such as multi-level splitting, will be necessary.

6.3 Sensitivity of IS Techniques

There are no general guidelines for determining the failure biasing parameters for simulations using importance sampling. The two main parameters we need to find are the *uniformization rate parameter* (β) and the *failure biasing probability* (P_{fb}). Both parameters are used in Algorithm 5 in Chapter 5, which represents the core of the uniformization-based importance sampling simulations presented in this chapter.

In this section, we determine the sensitivity of the P_{fb} parameter under the failure distributions, repair distributions and rate parameter (β) described in Section 6.1. We determine the stability of P_{fb} with probabilities ranging from 0.1 to 0.9. This analysis not only assures the stability of our techniques, it also provides us with stable P_{fb} to be used throughout this chapter.

6.3.1 Failure Biasing Probability Sensitivity

Each point in the following graphs represents 50000 simulation runs with mission time of 10 years. Error bars represent the relative error of each estimate. The disk failure distribution is exponential with rate 1/461386 and repairs are also exponential with rate 1/12. Here we explore the effect of the failure biasing probability on the resulting reliability estimate. Code Hamming distance, irregularity and size are varied to determine factors that may affect a reliability estimate.

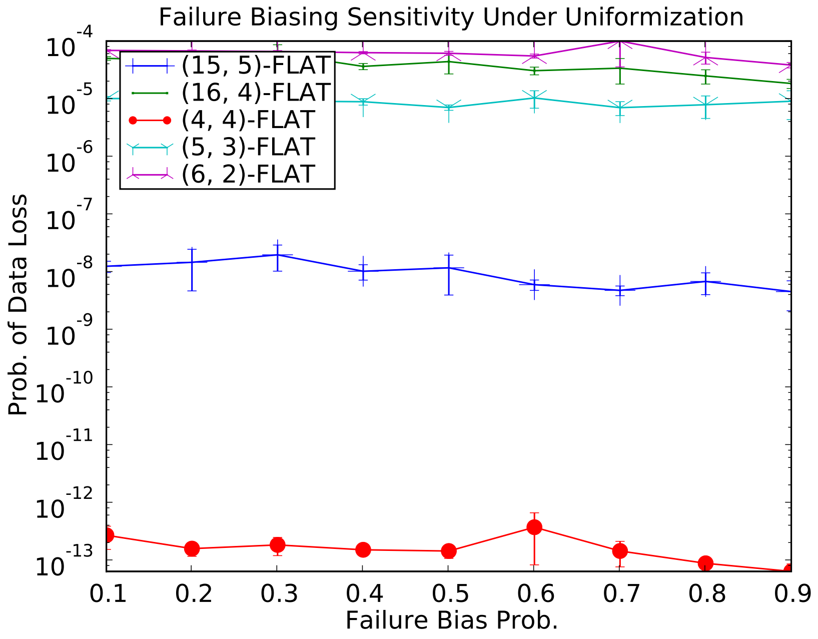
Figures 6.1-(a) and 6.1-(b) represent simulations that utilize failure biasing in a thinned (uniformization) process. Figure 6.1-(a) shows the probability of data loss in 10 years for a variety of FLATXOR-based codes across a failure biasing spectra from 0.1 to 0.9. Figure 6.1-(b) is the same, except for MDS codes. We find that, when compared to analytical estimates a failure biasing parameter between 0.2 and 0.4 appears to give accurate reliability estimates across all codes.

It is also quite clear that under uniformization, the reliability estimates of FLATXOR-based codes tend to be much more sensitive to the failure biasing parameter than the MDS codes. This is most likely due to the additional "coin flips" required when determining if a data loss event has occurred. As a consequence, evaluating a system with irregular fault tolerance under uniformization may require more iterations of the simulator before arriving at an accurate estimate.

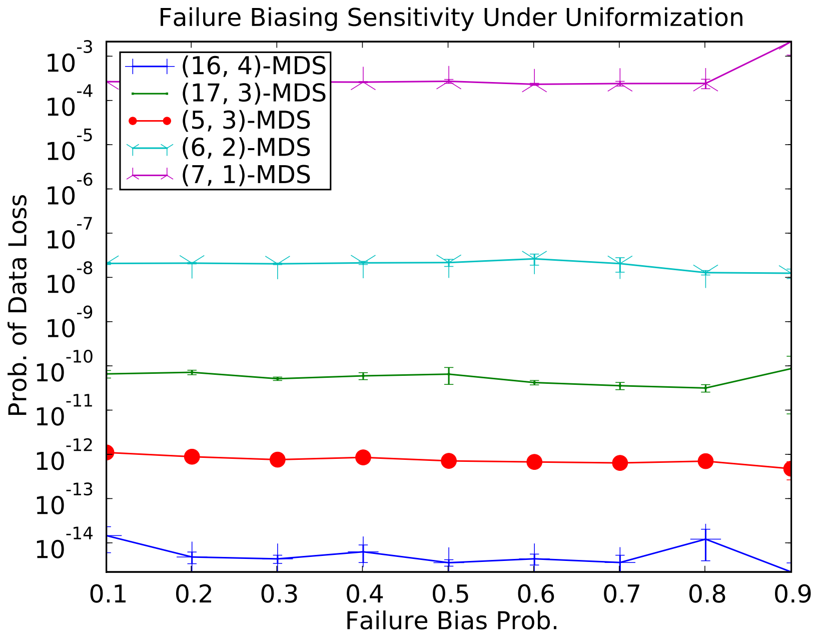
Figures 6.2-(a) and 6.2-(b) show the sensitivity of simulations under the direct sampling method. We find that the estimates in this case are much more stable than uniformization. This is because the holding times in each state are essentially drawn from the actual failure and repair distributions. While this type of simulation provides extremely accurate estimates under exponential failures and repairs, it may not work in general. In fact, the numerical methods package we used could not consistently invert the cumulative hazard function when simulating with the distributions published by Elerath and Pecht, thus the remainder of this evaluation relies solely on standard simulation and the uniformization-based balanced failure biasing.

Figures 6.3-(a) and 6.3-(b) show the sensitivity of a uniformization-based simulation under Elerath and Pecht's failure and repair distributions. Notice that changing the failure and repair distributions from exponential to the associated Weibull distributions results in a process that is very sensitive to the failure biasing parameter. As the figures show, all estimates are stable for parameters in the range 0.2 to 0.4. As the failure biasing parameter approaches 1, the resulting estimates slowly decrease in value. As with the estimates shown in Figure 6.3 irregular codes appear to be much more sensitive to the failure biasing parameter than the MDS codes. Additionally, the same behavior is apparent in highly fault-tolerant codes.

Unfortunately, unlike the simulations that rely on exponential distributions, we cannot verify the accuracy of each estimate using an analytical model. In the case of the simulations using non-exponential distributions, we use a failure biasing parameter in the range 0.2 to 0.4 and report the estimate with the lowest relative error. In addition to this heuristic, we can also verify certain estimates against standard simulation.

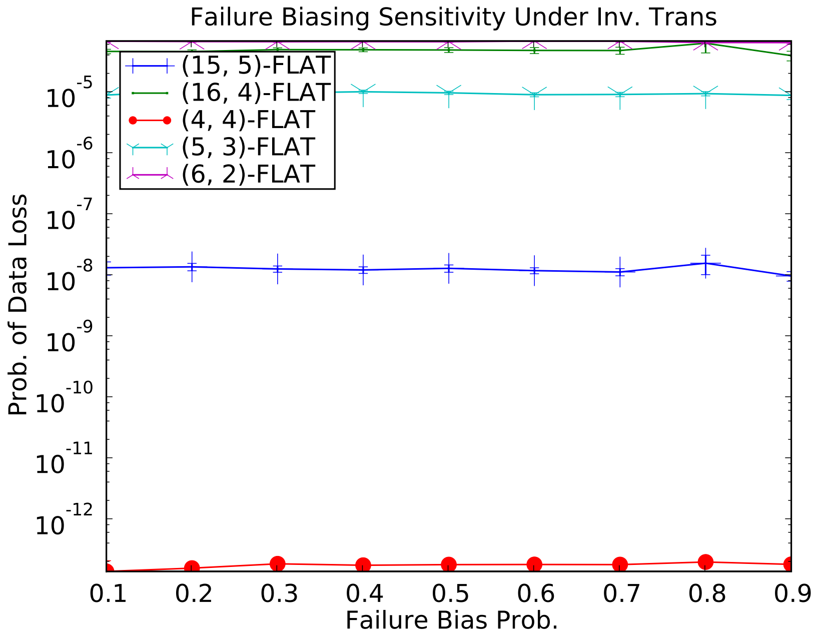


(a)

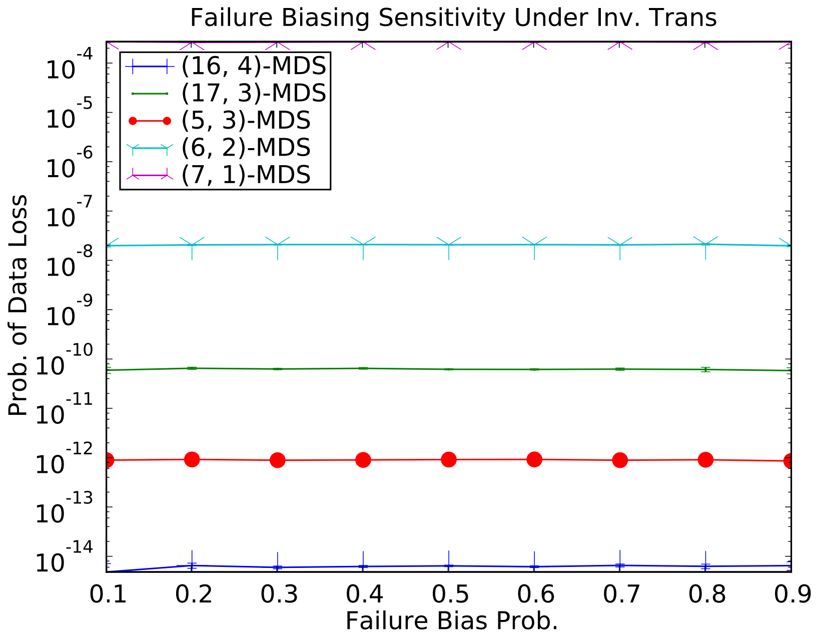


(b)

Figure 6.1: Sensitivity of failure biasing probabilities under uniformization with exponential failure and repair distributions.

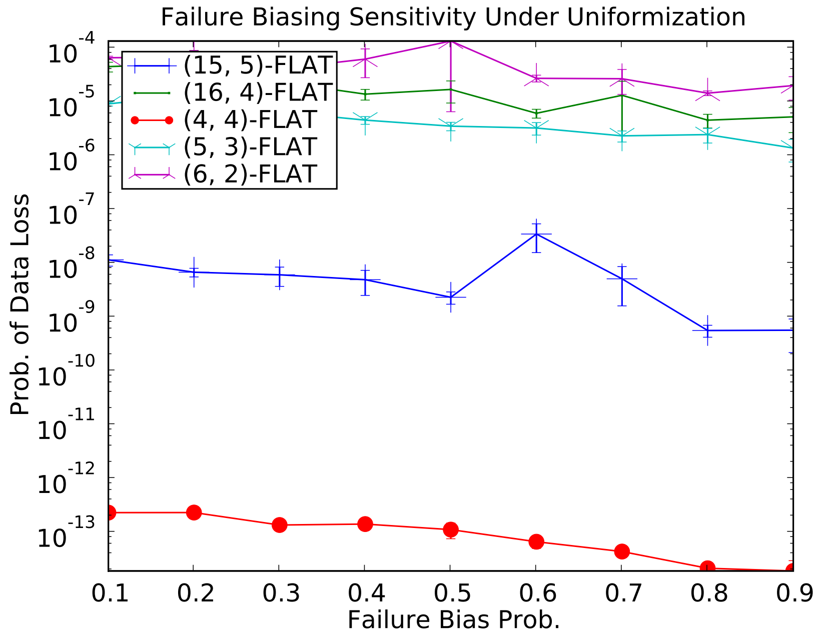


(a)

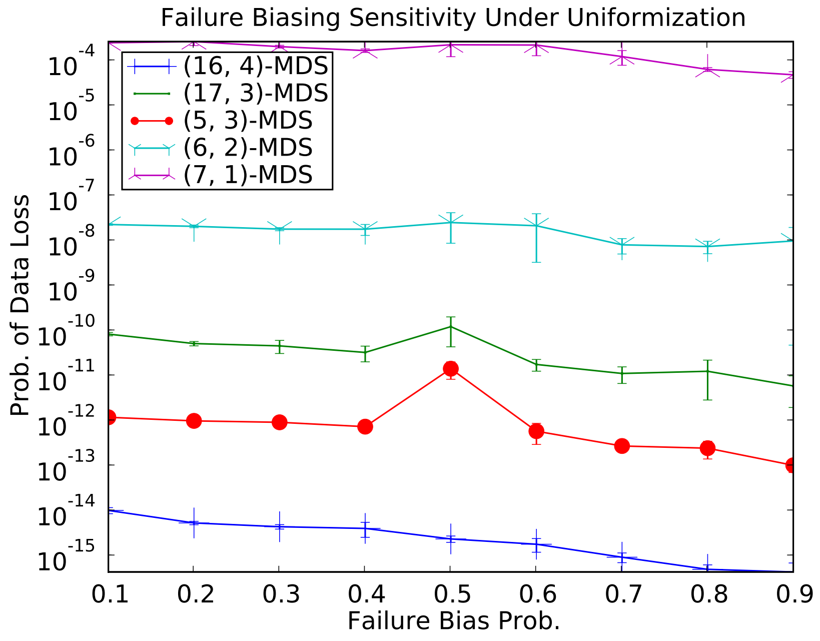


(b)

Figure 6.2: Sensitivity of failure biasing probabilities under inverse transform with exponential failure and repair distributions.



(a)



(b)

Figure 6.3: Sensitivity of failure biasing probabilities with Elerath and Pecht's failure and repair distributions.

6.4 Analysis of Failure and Repair Models

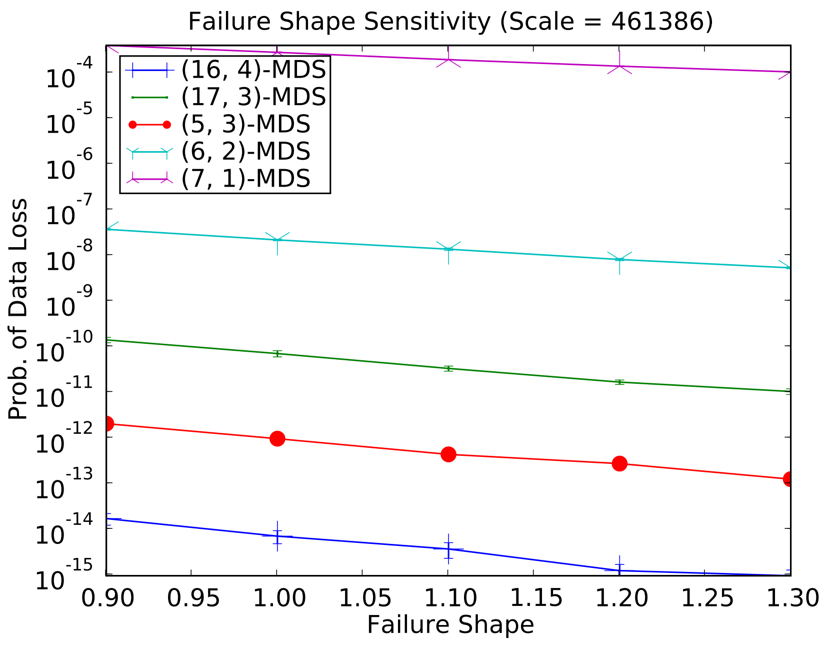
As we saw in Section 6.2, the efficiency of the HFR Simulator gives a system designer the ability to perform detailed trending and sensitivity analysis. In this section we explore the effect of various failure and rebuild models on system reliability. First, shape parameter of a Weibull distribution is varied to determine the effect of increasing (and decreasing) failure and repair rates on reliability. Second, the reliability estimates of Weibull distributed failure and repair times are compared to reliability estimates from an exponential distribution with the *same mean* to show the expected inaccuracy when the failure or repair rate is held constant.

The first part of this analysis is straightforward: we compare the relative reliability of systems having failure and repair characteristics drawn from different Weibull distributions. In short, determine the sensitivity of the Weilbull shape parameter for failures and repairs. The second part of this section is more subtle. Here we want to determine if the exclusion of time dependence in a model makes a difference. This is done by choosing a Weibull distribution with mean λ and using it as the failure or repair distribution in simulation. A system is also modeled as a Markov model with failures or repairs being exponentially distributed with mean λ. The output of each computation is compared. The difference between the reliability estimate will provide an approximation on how the constant failure or repair rate of an exponential distribution (Markov model) affects the reliability estimate.

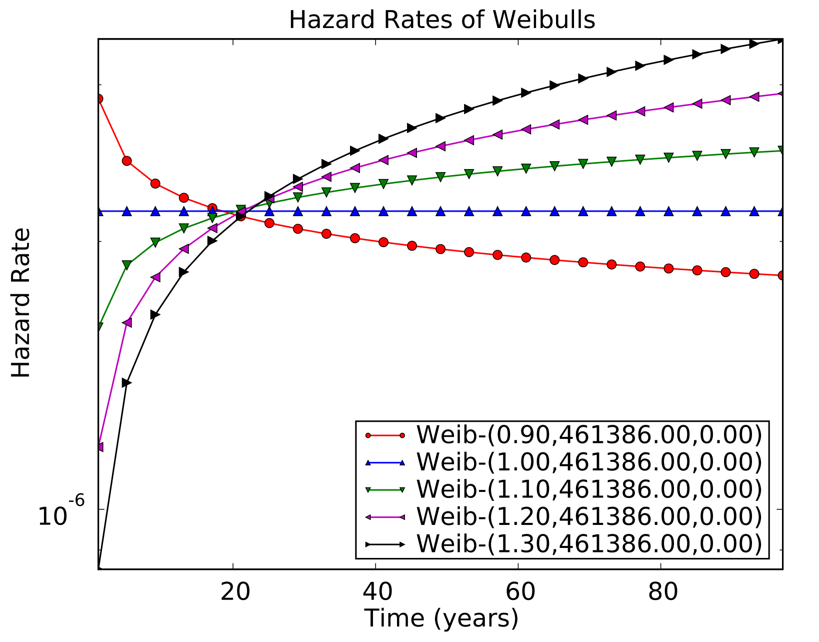
6.4.1 Failure and Repair Distribution Sensitivity Analysis

As pointed out earlier in Chapter 4, a primary concern with analytical models is their inability to handle time dependence. In this section, we explore the sensitivity of failure and repair distributions on reliability estimates. The spectrum of parameters is motivated by Elerath and Pecht's failure and repair distributions. The scale parameter is held constant (461386 for failures and 12 for repairs) and shape parameter is varied (0.90 to 1.30 for failures and 1.0 to 2.0 for repairs). Additionally, the location parameter is varied from 0.0 to 12.0 to better understand the impact of the location parameter on repairs.

Figure 6.4-(a) shows the effect of increasing failure shape on reliability of



(a)



(b)

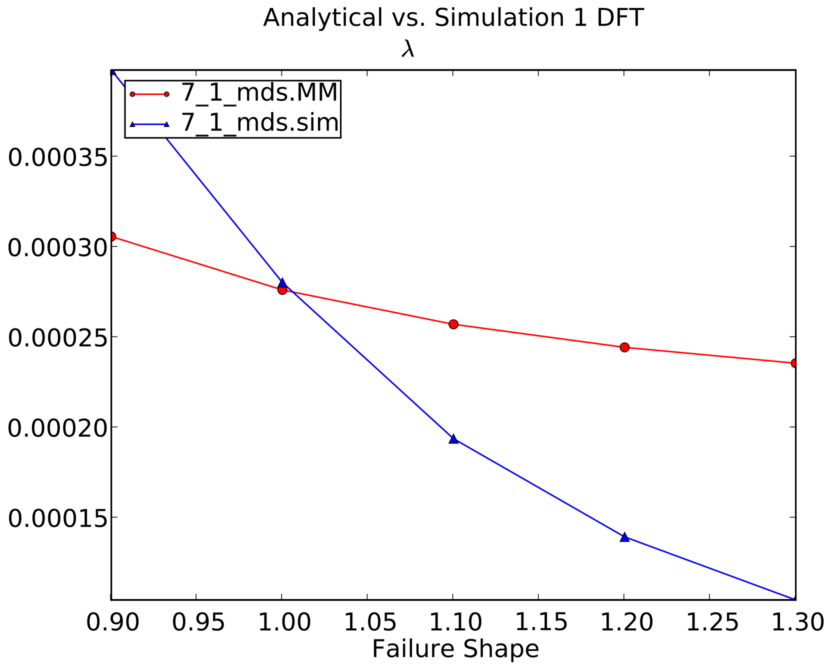
Figure 6.4: Effect of device failure distribution shape on system reliability.

(16,4)-MDS, (17,3)-MDS, (5,3)-MDS, (6,2)-MDS and (7,1)-MDS. We hold the scale at 461386 constant, the repair distribution constant with a repair scale of 12 and evaluate the reliability of each code with the following two-parameter Weibull distributions: Weibull(461386, 0.9), Weibull(461386, 1.0), Weibull(461386, 1.1), Weibull(461386, 1.2) and Weibull(461386, 1.3). The respective means of each failure distribution are: 485462, 461386, 445197, 434005 and 426125.

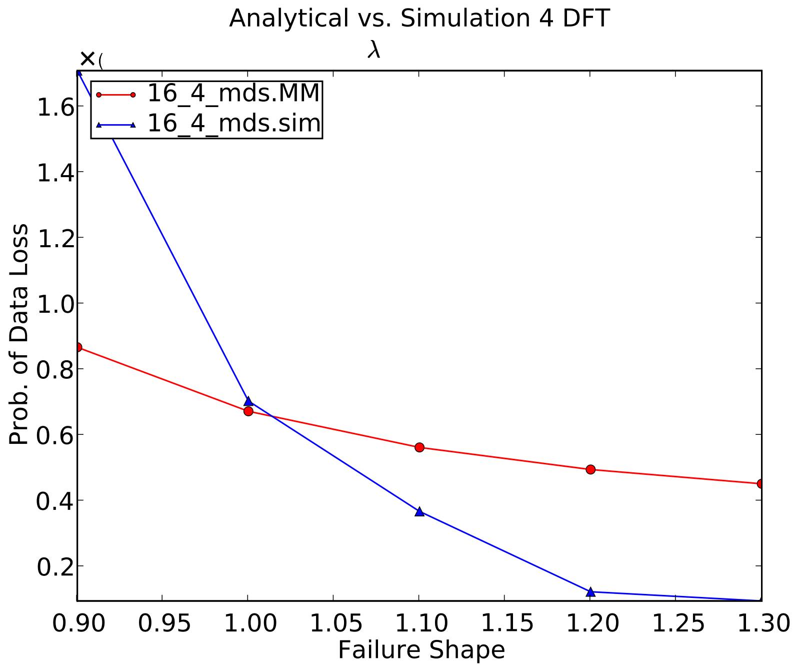
Figure 6.4-(b) shows the instantaneous failure rate as a function of time for each of the failure distributions. For distributions with a shape parameter less than 1, the failure rate starts high and decreases with time. The distribution with a shape parameter of 1 is an exponential distribution and has a constant failure rate. The failure rate for distributions with a shape parameter greater than 1 starts out low and increases with time. Note that for all distributions with a shape parameter not equal to 1, the failure rate will never reach that of the exponential distribution for the mission time of 10 years; thus, the shape parameter alone does not necessarily capture the effect of failure distribution on reliability.

As shown in Figure 6.4-(a) the underlying failure distribution can have a dramatic effect on system reliability. Looking across the spectrum of shapes, the estimated reliability can varies in some cases by more than an order of magnitude.

While Figure 6.4-(a) shows that the underlying failure distribution can have a dramatic effect on reliability, one may question the validity of a Markov model that aggregates the details of a complex failure distribution into a single mean (i.e. the shape parameter). Figures 6.5-(a) and 6.5-(b) show the reliability estimates taken from simulation of the actual Weibull failure distributions and a Markov model that assumes a constant failure rate taken as the mean of the Weibull distribution. Figure 6.5-(a) corresponds to the (7,1)-MDS code and Figure 6.5-(b) corresponds to the (16,4)-MDS code. As expected, the estimates for the failure distribution with a shape of 1 are close to equal, since the failure rate also remains constant in simulation. Time dependence matters when the shape parameter is not equal to 1 and we find that the estimates provided by the Markov model may be off by a factor of 2 or more depending on the failure distribution. This shows that in many cases, a time dependent failure distribution cannot be estimated simply by its mean. This is especially true when the mission time



(a)



(b)

Figure 6.5: Sensitivity of varied failure shape under simulation compared to an exponential with the same mean in a comparable Markov model.

of the simulation will not cover the entirety of the failure distribution.

A similar analysis was performed for repair distributions. The failure distribution is a Weibull distribution with scale parameter 461386 and shape parameter 1. The scale of the repair distribution is held at 12 and the shape parameter is varied from 1.0 to 2.0. As Figure 6.6-(a) shows, a comparable Markov model may accurately estimate reliability of a 1 disk fault-tolerant system simply using the mean of the actual failure distribution. As the fault tolerance increases, the shape parameter may have a larger effect on the estimated reliability, resulting in an inaccurate estimate provided by a Markov model. This behavior is shown in Figure 6.6-(b), where the reliability taken from a Markov model results in an estimate that can be a factor of two or more away from the actual (simulated) reliability.

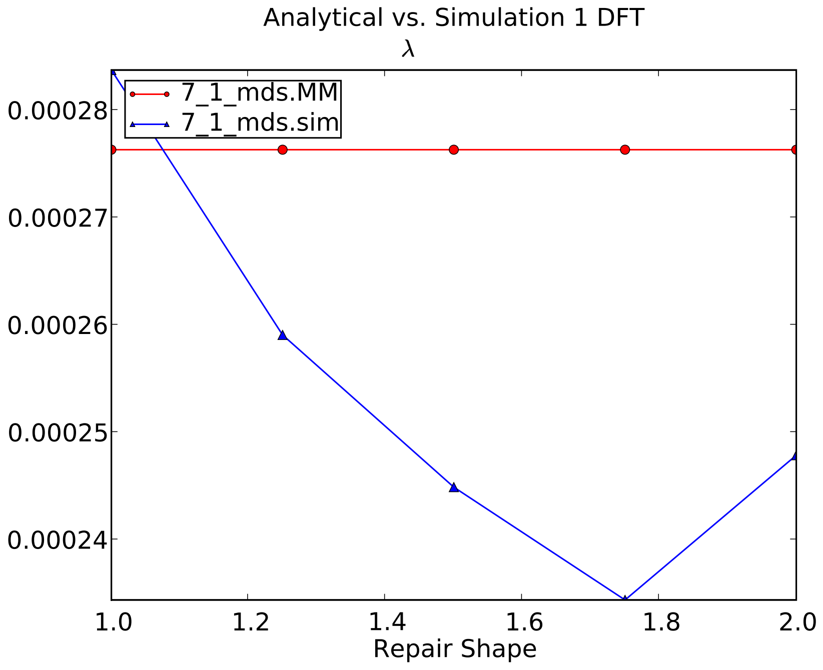
6.5 Erasure Code Sensitivity Analysis

Prior to the design and implementation of HFRS v.2, we performed a code-based sensitivity analysis. The HFRS v.1 simulator is used to perform an analysis of the sensitivity of four representative codes to varying disk rebuild rates, sector failure rates and sector scrub rates. The four codes we analyze are (36,12)-EVENODD, (8,8,2)-WEAVER, (42,12)-SPC, and (5,3)-FLAT. The goal of this sensitivity analysis is to determine which codes are most sensitive to disk rebuild time, sector failure rate, and scrub interval. The take away from this analysis is how different code constructions lead to sensitivities to different system parameters.

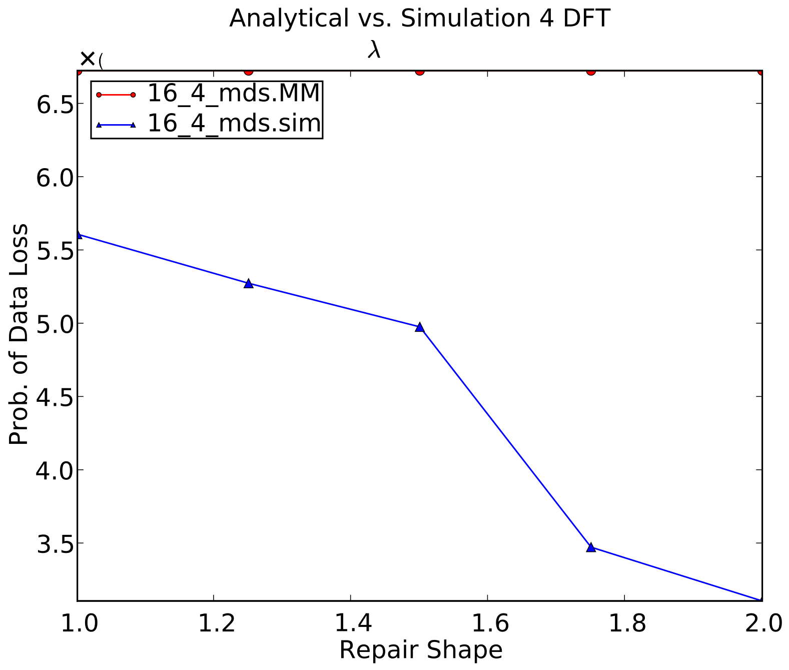
Here we use a normalized metric, called data-loss events per petabyte-year (DLE per PB-YEAR). DLE per PB-YEAR is measured as the inverse of the product of the capacity of the array (in petabytes) and the MTTDL (in years).

To speedup the analysis, we use a disk failure rate of $\lambda_d = 1/100,000$ hours. This rate is much higher than the disk failure rate used to generate the results in Table 6.2. Decreasing the rate drastically lowers the running time of the simulator, allowing us to quickly measure reliability across more configurations.

The disk failure rate remains constant throughout the analysis. A sensitivity analysis is performed for each of the sector failure, disk rebuild, and sector scrub rates. For each sensitivity analysis, all parameters are held constant except the one being



(a)



(b)

Figure 6.6: Sensitivity of varied repair shape under simulation compared to an exponential with the same mean in a comparable Markov model.

analyzed.

Figures 6.7, 6.8, and 6.9 show the results of these experiments. The left most data points in each figure is based the default parameters. In other words, the leftmost point is the same data point across all three graphs. The other data points are normalized with regard to the first data point. This is why the y-axis is unitless in each of these graphs. Given a rate or time period (x-axis), the reliability of a code can be calculated by multiplying the value for the initial data point by the corresponding y-value. The DLE per PB-YEAR values for the left most data points are listed in Table 6.9. Figures 6.7, 6.8, and 6.9 illustrate how all the codes exhibit linear sensitivity to each parameter, but that each code exhibits a different rate of change. Note that the linear increase corresponds to increased unreliability.

Figure 6.7 shows the effect of increasing disk rebuild time from 12 to 96 hours. The reliability of (5,3)-FLAT is essentially unaffected by change in rebuild time. This is because single-disk single-sector failures dominate the reliability of the code (there exists a minimal erasure of size 2). The reliability of all of the other codes is dominated by failure patterns involving double-disk failures and a single sector failure, and all are similarly affected by disk rebuild time.

Figure 6.8 illustrates the effect of increasing sector failure rate from 1/9259 to 1/925, while holding all other rates constant at their default values. As we see the SPC code maintains roughly the same level of reliability as sector failure rate increases. This is largely due to data loss being dominated by double-disk failures; the vertical parity protects against single-disk single-sector failures. We were surprised that the WEAVER code is more sensitive to increased sector failure rate than the EVENODD code. We expect this is because it is more likely that multiple sector failures exist at the time of a double-disk failure. This would affect the WEAVER code more because any sector failure causes data loss for the EVENODD code, whereas only specific sector failures cause data loss for the WEAVER code. Finally, Figure 6.9 shows the sensitivity to scrub interval. As expected, Figure 6.9 is heavily correlated with the results in Figure 6.8.

Overall, we find that rebuild rate has a smaller impact on reliability than factors affecting sector failures, such as the actual failure rates and scrub times.

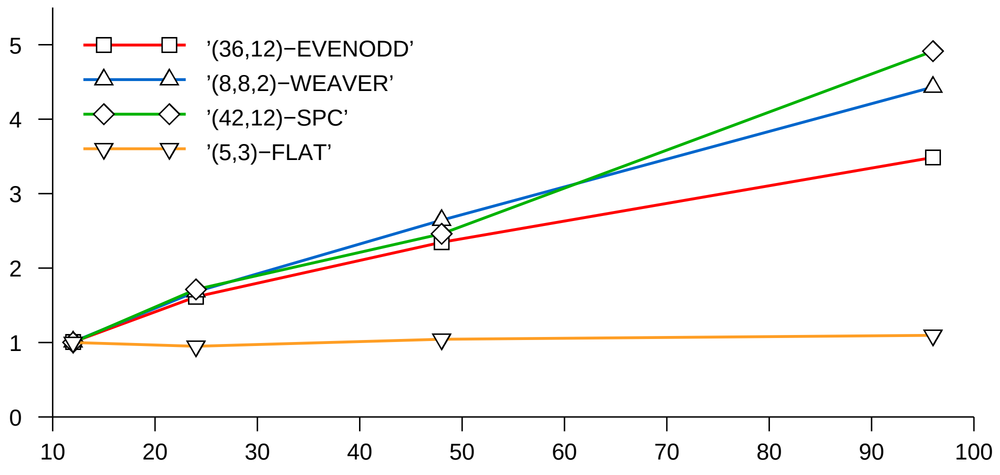


Figure 6.7: Normalized DLE per PB-YEAR vs. Expected disk rebuild time (hours)

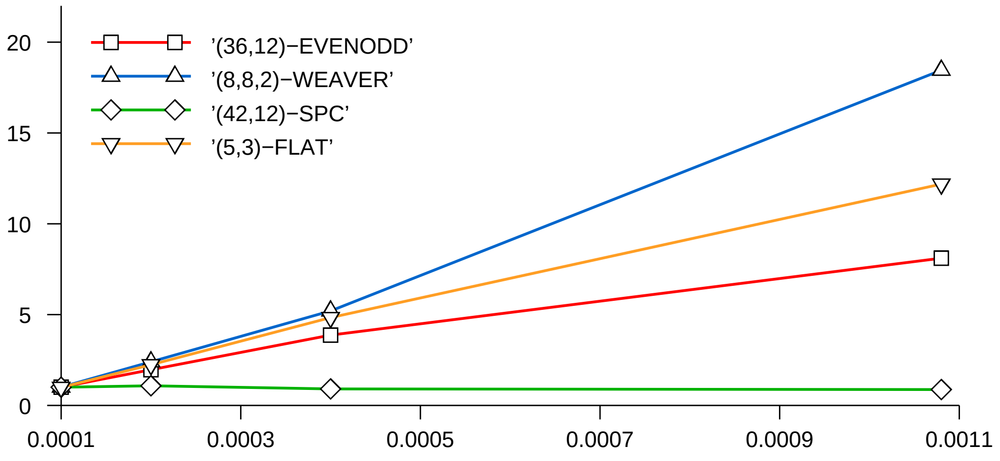


Figure 6.8: Normalized DLE per PB-YEAR vs. Sector failure rate (1/hours)

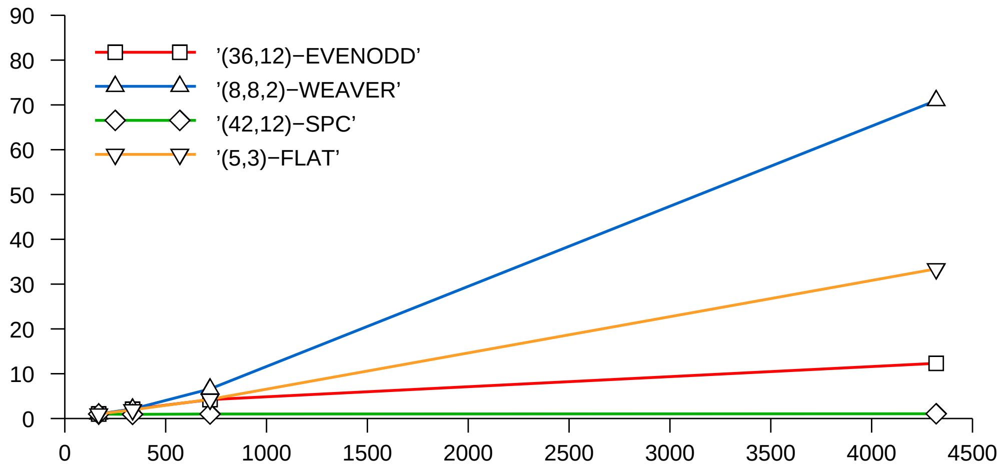


Figure 6.9: Normalized DLE per PB-YEAR vs. Expected time to scrub (hours)

code	**dle per pb-year**
(36,12)-EVENODD	0.0292
(42,12)-SPC	0.0510
(5,3)-FLAT	1.50
(8,8,2)-WEAVER	0.00313

Table 6.8: Base DLE per PB-YEAR of codes in sensitivity analysis.

code	**dle per pb-year**
(36,12)-EVENODD	0.0292
(42,12)-SPC	0.0510
(5,3)-FLAT	1.50
(8,8,2)-WEAVER	0.00313

Table 6.9: Base DLE per PB-YEAR of codes in sensitivity analysis.

rows	**rate**	SPC	SPC-NC
6	0.750	0.022	0.748
9	0.788	0.021	0.796
12	0.808	0.022	0.806
15	0.820	0.022	0.851
18	0.830	0.022	0.817

Table 6.10: DLE per PB-YEAR

6.5.1 SPC codes

We evaluated the impact of adding rows to the 8 disk SPC code. Increasing the number of rows increases the rate of the code. With a sufficiently large number of rows, the rate of an 8 disk SPC code would approach 0.875 (that of a comparable (7,1)-MDS setup). The results of our analysis are listed in Table 6.10. As the number of rows increases, the reliability remains almost constant. This is because the reliability is dominated by single disk single sector failures, regardless of the number of rows in the code. Amortizing the vertical parity over many rows, offers much greater reliability than (7,1)-MDS for a similar space efficiency. The reliability is premised on the independence of sector failures. Recent results suggest that this may be a poor assumption [3, 44]; others, however, have proposed extensions to SPC for tolerating bursts of sector failures [10].

In addition, we find that as we increase the amount of vertical parity in a SPC code, latent sector failures become extremely improbable. We compared simulation of an SPC code to a Markov model without sector failures in [58]. We found that as the amount of vertical parity is increased, the simulated MTTDL estimates approach the model with no sector failures. This observation is compelling in that scrub intervals can be widened substantially if a sufficient amount of vertical parity is used. These tradeoffs are left to future work.

code	**approx. MM**	**sim. approx**	**trad. MM**	**sim. trad.**	**sim. actual**
		latent sector errors			
(16,4)-MDS	9.23×10^{-13}	7.93×10^{-13}	7.00×10^{-12}	6.56×10^{-12}	3.03×10^{-12}
(17,3)-MDS	1.31×10^{-8}	1.36×10^{-8}	5.03×10^{-8}	4.61×10^{-8}	2.31×10^{-8}
(5,3)-MDS	1.92×10^{-10}	2.31×10^{-10}	7.59×10^{-10}	6.74×10^{-10}	3.48×10^{-10}
(6,2)-MDS	5.88×10^{-6}	5.49×10^{-6}	1.16×10^{-5}	1.06×10^{-5}	7.79×10^{-6}
(7,1)-MDS	7.18×10^{-2}	7.36×10^{-2}	7.18×10^{-2}	7.31×10^{-2}	7.51×10^{-2}
		ber errors			
(16,4)-MDS	3.00×10^{-12}	2.76×10^{-12}	2.06×10^{-11}	1.64×10^{-11}	1.07×10^{-11}
(17,3)-MDS	4.22×10^{-8}	3.98×10^{-8}	1.47×10^{-7}	1.43×10^{-7}	7.42×10^{-8}
(5,3)-MDS	6.32×10^{-10}	7.62×10^{-10}	2.43×10^{-9}	2.36×10^{-9}	1.25×10^{-9}
(6,2)-MDS	1.91×10^{-5}	2.03×10^{-5}	3.71×10^{-5}	3.87×10^{-5}	2.62×10^{-5}
(7,1)-MDS	2.09×10^{-1}	2.10×10^{-1}	2.09×10^{-1}	2.11×10^{-1}	2.09×10^{-1}

Table 6.11: A comparison of sector failure models.

6.6 Comparison of BER and Latent Sector Error Models

In Section 4.3, we presented the two dominating models for sector failures in storage systems. The first, called the BER model, derives the probability of a sector error on each disk from the device capacity and the bit-error rate. This models the occurrence of sector errors during the rebuild process. The second model, called the latent sector error model, assumes that sector errors may occur outside of the rebuild process and can be fixed by periodically scrubbing each disk and fixing the errors as they occur. Similar to the BER model, a probability is derived based on the underlying workload, the bit-error rate and the scrubbing frequency. The probability is used to determine if latent sector errors exist on any of the drives during the rebuild process.

In Section 4.3, we argued that current sector failure models do not account for the critical region when determining if a data loss event has occurred. The current models assume that the critical region encompasses the entire capacity of a device. We proposed two possible ways to accurately estimate the impact of sector errors on reliability by approximating the critical region. The first, assumes that at each successive failure, all on-going rebuilds have repaired half of their to-be-rebuilt capacity remaining since the last failure. The advantage of this approximation is the ability to incorporate it into Markov models. The resulting reliability is called the *approximate* estimate. The second and more robust estimate can only be done in simulation. The robust estimate keeps track of the rebuild time of the repair that is first to finish in the rebuild chain. This rebuild time is used to estimate the critically exposed sectors. The resulting reliability is called the *actual* estimate.

Table 6.11 shows the estimated reliabilities of each sector failure model for 5 MDS codes. We call the estimates based on processes that do not keep track of the critically exposed region *traditional*. Three main conclusions can be drawn from this table. First, the reliabilities estimates by the BER model and latent sector model can vary anywhere from a factor of 2 to an order of magnitude. In every case, the BER model estimates are always larger than the latent sector error model. Second, the traditional model may overestimate the probability of failure by an order of magnitude, as illustrates by the approximated critical region during the rebuild process of a 4 disk fault-tolerant code. Finally, as shown in the table, the expected "actual" probability of data loss falls somewhere between the traditional approach and the approximate estimate. The traditional estimate is roughly a factor of 2 from the expected "actual" probability.

The storage community has no yet reached a consensus on how to correctly model sector errors. We believe the actual data loss probability will be between the latent sector error estimates and the BER estimates, given the critical region is approximated by the "actual" estimate.

6.7 Sensitivity of Bookkeeping Structures

The HFR Simulator supports three bookkeeping structures that are used to determine if a data loss event has occurred. The structures, covered in Section 5.3.3, are matrix rank, MEL and FTM (or FTV). In this section, we evaluate the sensitivity of these structures to both simulation performance and estimated reliability.

Table 6.12 shows the sensitivity of each structure on the estimated probability of data loss across 5 flat XOR-based codes. The MEL and a rank test are essentially

	FTV		MEL		
code	**failure prob.**	**rel. err.**	**failure prob.**	**rel. err.**	**analytical**
(4,4)-FLAT	2.07×10^{-13}	9.32	1.93×10^{-13}	9.60	1.87×10^{-13}
(15,5)-FLAT	1.21×10^{-8}	11.03	1.09×10^{-8}	11.77	1.23×10^{-8}
(16,4)-FLAT	5.15×10^{-5}	6.60	4.47×10^{-5}	7.69	4.94×10^{-5}
(5,3)-FLAT	9.74×10^{-6}	6.78	9.17×10^{-6}	22.23	9.88×10^{-6}
(6,2)-FLAT	6.87×10^{-5}	2.48	7.05×10^{-5}	2.49	6.91×10^{-5}

Table 6.12: Reliability comparison of FTV and MEL

equivalent methods of determining data loss; thus, we only evaluate the MEL. We built an analytic model based on the FTV for each code (similar to Hafner and Rao), which serves as a comparison point. All of the estimates given by the FTV and MEL are in agreement, which suggests that they can be used interchangeably.

Figure 6.10 compares the performance of each bookkeeping structure for two flat XOR-based codes: (15,5)-FLAT and (5,3)-FLAT. The (5,3)-FLAT code has a 5×8 generator matrix and an MEL of length 22. The (15,5)-FLAT code has a 15×20 generator matrix and an MEL of length 1540. At each failure event a data loss event is detected by either transforming an updated generator matrix into reduced row echelon form (via elementary operations), performing a brute force search of the MEL or looking up the appropriate probability in the FTV and drawing a uniform random number.

Figure 6.10 illustrates the effect of bookkeeping structure on performance. In general, the rank check will result in the worst performance for a given code. When the MEL for a code is relatively large, bookkeeping using the MEL can lead to very poor performance. In general, bookkeeping using the FTV will result in the best performance regardless of code size.

6.8 Apples-to-Apples Comparison of Erasure Codes

In this section we perform an apples-to-apples comparison of the erasure codes listed in Table 6.2. The resulting failure probabilities cover a range that is roughly 10 orders of magnitude. A comprehensive comparison of different codes would include

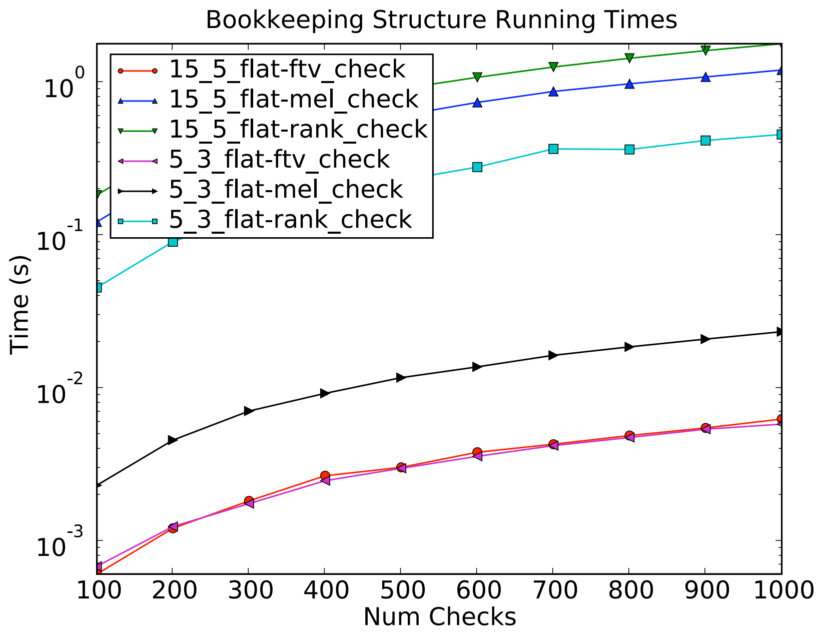


Figure 6.10: Performance comparison of FTV, MEL and matrix rank for (15,5)-FLAT and (5,3)-FLAT. *Num checks* represents the number of lookups (for FTV and MEL) or matrix canonical form transformations (for rank check) run to check for data loss.

dimensions other than reliability, such as performance and cost. Since we are mainly interested in reliability, we cannot draw sweeping conclusions about erasure codes from this analysis. The main take-away in this analysis is the fact that we can evaluate the reliability of different erasure codes under the same framework. Previous reliability simulators and models do not have the ability to such analysis.

The analysis is split into two parts: accuracy/performance and reliability. First, the relative accuracy and performance of each simulation technique is compared across the codes. Finally, we compare the reliability of different codes in similar classes: codes with the same Hamming distance.

6.8.1 Accuracy and Performance

The data loss probabilities shown in Table 6.13 represent the most accurate estimates where the simulations under importance sampling were run for $250K$ iterations and the standard simulations were run for 10M iterations. It turns out that in most cases the relative error of each simulation is around or below 20%. A relative error of 100% represents a factor of 2 from the estimate; thus, the resulting confidence interval is well below a factor of two on either side. One interesting exception is the (5,3)-FLAT code, which after 250K iterations has a relative error of almost 50% when using balanced failure biasing. In this case, an accurate reliability estimate can be obtained much faster through standard simulation.

Overall, any code with a Hamming distance of 2 can be efficiently and accurately estimated using standard simulation. Under importance sampling, the 8 disk configurations take roughly 1000 seconds, while the 20 disk configurations take anywhere from roughly 6000 to 8000 seconds. We find that at or beyond 2 disk fault tolerance, it is most likely more efficient to use importance sampling, though the estimates may not be as accurate as standard simulation.

6.8.2 Reliability

Consider all of the codes with a Hamming distance of 2: (7,1)-MDS, (5,3)-FLAT, (6,2)-FLAT, (16,4)-FLAT and (8,8,1)-WEAVER. In most cases, we find that as

code	sim. type	prob. prob.	rel. err. (%)	num iter.	run time (s)
(16,4)-MDS	BFBUNIF	1.19×10^{-12}	19.85	250K	2692.57
	SIMREG	—	—	—	
(17,3)-MDS	BFBUNIF	1.16×10^{-8}	17.76	250K	1751.64
	SIMREG	—	—	—	
(5,3)-MDS	BFBUNIF	1.92×10^{-10}	12.88	250K	490.13
	SIMREG	—	—	—	
(6,2)-MDS	BFBUNIF	8.26×10^{-6}	12.70	250K	957.24
	SIMREG	5.90×10^{-6}	21.42	10M	3517.30
(7,1)-MDS	BFBUNIF	4.46×10^{-2}	7.57	100K	192.01
	SIMREG	5.85×10^{-2}	2.09	100K	137.89
(36,12)-EVENODD	BFBUNIF	8.51×10^{-6}	13.11	250K	977.35
	SIMREG	4.20×10^{-6}	25.38	10M	3603.11
(36,12)-RDP	BFBUNIF	8.32×10^{-6}	11.00	250K	991.90
	SIMREG	6.40×10^{-6}	20.56	10M	3585.73
(35,14)-XCODE	BFBUNIF	5.39×10^{-6}	13.85	250K	791.21
	SIMREG	5.30×10^{-6}	22.60	10M	3054.41
(8,8,1)-WEAVER	BFBUNIF	6.90×10^{-3}	11.56	250K	825.21
	SIMREG	8.54×10^{-3}	5.61	100K	70.72
(8,8,2)-WEAVER	BFBUNIF	8.87×10^{-7}	23.21	250K	1214.07
	SIMREG	8.00×10^{-7}	58.16	10M	3528.44
(8,8,3)-WEAVER	BFBUNIF	6.27×10^{-11}	21.66	250K	1441.06
	SIMREG	—	—	—	
(42,12)-SPC	BFBUNIF	2.25×10^{-4}	11.40	100K	205.10
	SIMREG	2.56×10^{-4}	3.25	10M	7019.48
(6,2)-FLAT	BFBUNIF	1.27×10^{-2}	18.65	100K	304.56
	SIMREG	1.53×10^{-2}	4.17	100K	71.28
(5,3)-FLAT	BFBUNIF	2.54×10^{-3}	47.80	250K	1324.07
	SIMREG	2.44×10^{-3}	10.52	100K	71.84
(4,4)-FLAT	BFBUNIF	1.15×10^{-10}	24.30	250K	1897.87
	SIMREG	—	—	—	
(15,5)-FLAT	BFBUNIF	2.60×10^{-6}	28.62	250K	8185.62
	SIMREG	2.80×10^{-6}	31.09	10M	10875.06
(16,4)-FLAT	BFBUNIF	6.65×10^{-3}	32.20	250K	5694.39
	SIMREG	1.08×10^{-2}	4.97	100K	339.79

Table 6.13: Apples-to-apples comparison of erasure codes.

the coding rate increases, probability of data loss decreases. For example, this holds true for (7,1)-MDS, (5,3)-FLAT, (6,2)-FLAT and (16,4)-FLAT. This does not hold true when comparing the reliability of (8,8,1)-WEAVER and (5,3)-FLAT. Even though the rate of (8,8,1)-WEAVER is lower than (5,3)-FLAT, it does not provide better reliability. The (8,8,1)-WEAVER is constructed to offer regularity and symmetry for the sake of performance and manageability, which reduces reliability.

The (42,12)-SPC has a “disk” Hamming distance of 2 (i.e. can tolerate any single, whole-disk failure). Its reliability is significantly better than the other codes with Hamming distance 2. This is mostly due to the fact that the symbol-wise Hamming distance is 4, which protects against sector errors. This is consistent with the HFRS v.1 code sensitivity analysis performed in Section 6.5.

Consider the codes with a Hamming distance of 3: (6,2)-MDS, (36,12)-EVENODD, (36,12)-RDP, (35,14)-XCODE, (8,8,2)-WEAVER and (15,5)-FLAT. The (6,2)-MDS, (36,12)-EVENODD and (36,12)-RDP have the same rate and similar reliability. This is expected because the codes only differ in the tolerance of multiple sector failures in the same strip. It turns out that the most probable path to failure is double-disk with a single sector, which accounts for the most probable data loss events for these codes. (15,5)-FLAT has the same rate as (6,2)-MDS and a much better reliability in spite of covering 20 disks. As a rule of thumb, we believe that an irregular code with the same Hamming distance and rate as the comparable MDS code will most likely lead to a higher reliability, given it the code does not cover too many disks. (35,14)-XCODE has a slightly lower rate than the (6,2)-MDS code, and is slightly more reliable.

The (8,8,2)-WEAVER code is roughly a full order of magnitude more reliable than most of the other codes with a Hamming distance of 3. The dominant cause of data loss for these codes is a double disk in conjunction with a single sector failure. WEAVER codes offer better reliability because only one subset of double disk failures put it into a critical state, where an additional sector failure leads to data loss. For MDS codes, parity-check array codes, every double-disk single-sector failure combination leads to data loss.

Now, consider the codes with a Hamming distance of 4: (17,3)-MDS, (5,3)-MDS, (8,8,3)-WEAVER and (4,4)-FLAT. It is apparent that the lower rate codes have a higher reliability. The fact that (17,3)-MDS covers 20 disks and requires at least 17 disks for recovery leads to a reliability that is almost 2 orders of magnitude worse than (5,3)-MDS. This is most likely due to the higher probability of finding a sector failure in the critical mode when 17 disks are involved in the rebuild operation. Both the (4,4)-FLAT and the (8,8,3)-WEAVER result in reliabilities that are better than the MDS codes. Again, this is in agreement with the argument that lower rate tends to result in higher reliability.

There is a single code with Hamming distance 5: (16,4)-MDS. This code was mainly chosen for use in the sensitivity analysis earlier in this chapter and as expected provides the lowest probability of failure. Looking at the differences between the MDS codes with the same number of disks, we find that adding an additional parity disk leads to an increase in reliability by roughly 4 orders of magnitude.

We are hesitant to draw sweeping conclusions from the reliability results in Table 6.13. We believe that this "apples-to-apples" comparison of reliability of different erasure codes is the most extensive to date, and is a significant contribution. Even though we believe that Elerath and Pecht's model for failures and recoveries is the most compelling published model, many recent results suggest that a better model of disk failure and recovery is needed [4, 55, 49, 3, 44, 14]. Once such models become available we are confident that we can use them to refine the analysis we have done thus far.

6.9 Summary

The primary focus of this chapter was to validate the HFR Simulator and show that it can be used to perform a wide range of reliability analysis on erasure-coded storage systems. In the process, we have evaluated the sensitivity of importance sampling parameters under real-world failure characteristics, sensitivity of failure (or repair) distribution choice, erasure code sensitivity, the sensitivity of sector failure models, performance of the HFR Simulator bookkeeping structures and a wide-range of erasure codes in an apples-to-apples fashion.

A summary of the major findings of this evaluation is given in itemized form below.

- Importance sampling is very accurate for 1-4 disk fault-tolerant codes modeled by a Markovian process.

- Importance sampling is possible and reasonably accurate for 1-4 disk fault-tolerant codes modeled by a non-Markovian process. We found that importance sampling starts to break down around 4-disk fault tolerant systems. It was possible to achieve good estimates at 4-disk fault tolerance, but it took many trials.
- Given the device failure (or repair) distribution is non-Exponential, assuming a constant failure (or repair) rate derived from the mean can lead to inaccurate estimates. This implies that the use of Markov models may lead to inaccurate results.
- Current sector error models do not provide accurate reliability estimates. This is mostly due to not keeping track of the critical region during rebuild. Additionally, it is unclear which sector failure model is correct. The BER and latent sector error models result in different reliabilities; combining the two will most likely lead to a more accurate model.
- In general, the fault-tolerance vector (FTV) provides accurate and efficient book-keeping of XOR-based codes.
- The HFR Simulator is the only existing framework that can be used to compare the reliability of different codes in an apples-to-apples fashion.
- Typically, lower rate codes lead to better reliability. While this is true in most cases, structural differences between codes can provide a counterexample. For instance, a (6,2)-MDS code is typically more reliable than a mirrored configuration. This implies that density (number of code symbols per parity equation) will also have an effect on reliability.
- An irregular code with the same Hamming distance and rate as an MDS code will most likely have better reliability, given it does not cover too many devices. This was apparent when comparing a (15,5)-FLAT code to the (6,2)-MDS code.
- Structurally different codes may exhibit their own unique sensitivity to rebuild time and sector failure characteristics.

- Simple product codes (SPC) are generally unaffected by an increasing sector failure rate (or decreasing scrub rate).

Chapter 7

Layout of Erasure-Encoded Fragments Across Heterogeneous Devices

Erasure codes such as replication, RAID 5, and Reed-Solomon codes, are the means by which storage systems are typically made reliable. Reed-Solomon codes provide the best trade off between fault tolerance and space-efficiency, but are computationally the most demanding type of erasure code. In addition to these traditional erasure codes, there are a number of proposals for novel erasure codes that exclusively use XOR operations to generate redundancy (e.g., [21, 22, 65]). Such XOR-based codes are computationally more efficient than Reed-Solomon codes, but offer a non-uniform trade off between performance, space-efficiency, and fault tolerance.

Methods to evaluate the space-efficiency and performance trade off for XOR-based codes are fairly well understood [23, 30, 51]. However, XOR-based erasure codes exhibit irregular fault tolerance: some subsets of failed storage devices of a given size lead to data loss, whereas other subsets of failed storage devices of the same size are tolerated. There have been many recent advances in understanding the irregular fault tolerance [24, 65] and concomitant reliability [52, 25] of XOR-based codes. However, all of these advances assume a *homogeneous* set of storage devices that all fail and recover at similar rates.

The contributions of this work are fourfold. First, we identify the *redundancy placement problem*, a novel reliability problem in storage systems: in a storage system comprised of a *heterogeneous* set of storage devices with known failure and recovery rates, how should erasure-coded symbols be placed to maximize reliability? The redundancy placement problem is trivial for Reed-Solomon style codes because such codes exhibit regular fault tolerance—all sets of device failures of some size lead to data loss—so all placements have the same reliability. For XOR-based codes, however, the redundancy placement problem is non-trivial to solve because some failure sets are larger than others and may involve devices of different reliability. Second, we propose a simple analytic model, the Relative MTTDL Estimate (RME), that allows the relative reliability of different placements to be compared in a computationally efficient manner. Third, we propose two redundancy placement algorithms that use the structure of the XOR-based erasure code and the RME to determine a placement that maximizes (estimated) reliability. Fourth, we empirically demonstrate, via simulation, that the RME correctly orders different placements with regard to their reliability, and that the redundancy placement algorithms identify placements that maximize reliability. The empirical analysis relies heavily on the techniques presented in Chapter 4. The HFRS v.1 implementation is used throughout this chapter.

7.1 Redundancy Placement Algorithms

We have developed two redundancy placement algorithms that identify placements of erasure-coded symbols on heterogeneous storage devices with known failure and repair rates which maximize reliability. One redundancy placement algorithm is based on brute force computation and the other is based on simulated annealing.

More formally, let S be the set of symbols in the erasure code and D be the *configuration* (set of heterogeneous devices). For a code with n symbols, $S = \{s_0, \ldots, s_{n-1}\}$ and $D = \{d_0, \ldots, d_{n-1}\}$. A placement, ρ, is a bijective function that uniquely maps each symbol in the erasure code to a single device: $\rho : S \leftrightarrow D$. The goal of the redundancy placement algorithms is therefore to find a placement ρ that maximizes reliability.

7.1.1 Relative MTTDL Estimate

We now introduce the simple analytic model that underlies both redundancy placement algorithms: the Relative MTTDL Estimates (RME). The RME can be used to compare the reliability of different placements. It is constructed to correlate with the expected MTTDL (e.g., as determined by HFRS v.1), but it does not accurately estimate the MTTDL. The RME can be used to compare the relative merit of different placements, but not to determine if some placement meets a specific reliability requirement.

At a high level, the RME is the inverse of an estimate of the expected unavailability of a given placement. It is based on the MEL and a simple analytic device model. The MEL is a set of sets of erasure-coded symbols, i.e., $\text{MEL} \subset 2^S$, with the property that $\neg\exists f, f' \in \text{MEL} : f \subseteq f'$, and so is a concise, exact description of irregular fault tolerance. Let $\mathrm{u}(d)$ be the expected unavailability of device d. To calculate $\mathrm{u}(d)$, the MTTR of d is simply divided by its MTTF. This analytic model ignores sector failures and scrubbing, as well as the distribution of the device failures and repairs. The RME is calculated via the following function of the redundancy placement ρ, device unavailability u, and MEL:

$$\text{RME} = \left(\sum_{f \in \text{MEL}} \prod_{s \in f} \mathrm{u}(\rho(s)) \right)^{-1}.$$

The sum of products is inverted because RME values are values that should be maximized to improve reliability, just like MTTDL values.

The RME for the (4,4)-RAID 10 code with MEL $\{(s_0, s_4), (s_1, s_5), (s_2, s_6), (s_3, s_7)\}$ is as follows:

$$\begin{aligned} \text{RME} &= (\mathrm{u}(s_0)\mathrm{u}(s_4) + \mathrm{u}(s_1)\mathrm{u}(s_5) + \\ &\qquad \mathrm{u}(s_2)\mathrm{u}(s_6) + \mathrm{u}(s_3)\mathrm{u}(s_7))^{-1}. \end{aligned}$$

Consider a configuration in which the first 4 devices have expected device unavailability of 1.2×10^{-4} and the second 4 devices have expected device unavailability of 2.4×10^{-5}. Note that the more reliable a device is, the lower its device unavailability number, so the first 4 devices are less reliable than the second 4 devices in this configuration. Now consider two distinct placements. In the first placement, the first 4 symbols are placed on the first 4 devices, and the second 4 symbols are placed on the second 4 devices, and so the $\text{RME} = 86.8 \times 10^6$. In the second placement, the "odd symbols" (i.e., s_1,s_3,s_5, and s_7) are placed on the first 4 devices, and the "even symbols" on the second 4 devices, and so the $\text{RME} = 33.4 \times 10^6$. The first placement splits the pair of replicated symbols that occur in each minimal erasure so that one is placed on the less reliable device and the other on the more reliable device. In contrast, the second placement places all of the symbols from two minimal erasures (the "odd symbols") on the less reliable devices, which, intuitively, is a less reliable placement. The RME values for the two placements correctly order them regarding our intuition about their relative reliability; this intuition is confirmed via simulation, as described in Section 7.2.1.

The decision to use a simple analytic model is based on several reasons. First, the simplicity of the analytic device model permits efficient evaluation of the RME and so permits orders of magnitude more distinct placements to be evaluated than simulation methods in the same period of time. Second, the model only has to produce an RME that accurately orders different sets of device failures according to the likelihood that they contribute to data loss. The product of expected device unavailability accomplishes this task. Third, in a system with any redundancy, sector failures alone do not cause data loss; only multiple disk failures, or disk failures in conjunction with sector failures lead to data loss. Thus, the simple analytic model only needs to capture the reliability affects of disk failures. We discuss the effect of sector failures further in Section 7.2.2.

7.1.2 Brute Force Algorithm

The brute force redundancy placement (BF-RP) algorithm evaluates the RME for all possible placement and identifies the placement with the largest RME as the best placement. The RME is a simple equation that can be evaluated efficiently. To calculate an RME value requires $|\text{MEL}|$ additions and less than $m \times |\text{MEL}|$ multiplications. Consider the calculation of the RME for (4,4)-RAID 10 given above. It required four additions because $|\text{MEL}| = 4$, and four multiplications because each of the four minimal erasures consisted of exactly two symbols. Since all minimal erasures consist of m or fewer symbols, each such product requires $m - 1$ or fewer multiplications.

For a code with n symbols, there are $n!$ possible placements to evaluate. Given the efficiency of the RME calculation, it is feasible to evaluate the RME for every possible placement for small codes. For example, in Section 7.2 the BF-RP algorithm is used to find the best placement for some codes with $n = 12$. Each such execution of the BF-RP performs $12! = 479001600$ RME calculations to determine the best placement.

7.1.3 Simulated Annealing Algorithm

For large codes, the factorial number of distinct placements make it is infeasible to apply the BF-RP algorithm. The best placement for a code maximizes the RME value. Therefore, the problem of finding the best placement can be understood as an optimization problem. Unfortunately, the nonlinear structure of the RME equation—all of the terms in the summation are products of variables to be assigned via the optimization—precludes linear optimization techniques.

Fortunately, there are many non-linear optimization techniques. An approach that requires little work, in terms of formulating constraint equations, is simulated annealing [32]. This made simulated annealing, a stochastic optimization technique, appealing as the first optimization approach for us to evaluate. Simulated annealing uses randomization to find a solution; however, there is a chance that the solution found is not globally optimal.

The simulated annealing redundancy placement (SA-RP) algorithm takes an MEL and a configuration of devices as input. The SA-RP algorithm is initialized with a randomly selected placement. Each *step* in SA-RP is based on a random number of random swaps of mappings in the current placement. As the algorithm proceeds, the number of random swaps performed at each step decreases. This is the manner in which we capture the "cooling" aspect of simulated annealing, in which randomness is reduced over time so that some locally optimal placement is settled upon. In SA-RP, we include parameters to backtrack if a step that decreased the RME does not, after some number of additional steps, lead to a larger RME value. The SA-RP algorithm is invoked multiple times, while keeping track of the best RME value found over different invocations. Because each invocation is initialized with a different random placement, repeated invocations finds distinct locally maximal placements (RME values).

Unfortunately, simulated annealing does not lend itself to many practical rigorous statements about the quality of solution found. However, our empirical evidence to this point indicates that the SA-RP algorithm quickly produces good solutions.

7.2 Evaluation

To evaluate the BF-RP and SA-RP algorithms, we consider configurations that have devices with failure models between two bounds. The first device failure model is based on that used by Elerath and Pecht (cf. Table 2 in [14]). Disk failures are distributed according to a Weibull distribution with parameters $\gamma = 0$, $\eta = 500000$, and $\beta = 1.12$. (Note that we "rounded up" the η parameter of 461386 hours used by Elerath and Pecht). Disk recoveries are distributed according to a Weibull distribution with $\gamma = 6$, $\eta = 12$, and $\beta = 2$. We refer to the first device as the 500k device because its expected MTTF is 500 thousand hours. The 500k device is the most reliable device we consider in the evaluation. We refer to the least reliable device as the 100k device. The 100k device differs from the 500k device only in its MTTF: $\eta = 100000$ instead of $\eta = 500000$. To calculate the RME, only the MTTF for disk failure and the MTTR for recovery is used. The HFR Simulator (HFRS v.1) uses the specified Weibull distributions to simulate the MTTDL. In some simulations, the device failure models include latent sector failure and scrubbing: the latent sector failures per disk are distributed according to an exponential distribution with mean 9259, and are scrubbed (recovered) according to a Weibull distribution with $\gamma = 6$, $\eta = 168$, and $\beta = 3$.

There are two types of heterogeneous configurations we evaluate. *Bimodal configurations* consist of only two types of devices: 100k devices and 500k devices. For example, an 8 disk 3-bimodal configuration consists of 3 100k devices and 5 500k devices. *Uniform configurations* consist of one 100k device and one 500k device; the remaining devices have MTTF values uniformly distributed between $\eta = 100000$ and $\eta = 500000$. For example, an 8 disk uniform configuration consists of one device with each of the following η values: 100000, 157000, 214000, 271000, 328000, 385000, 442000, 500000. We evaluate 8, 12 and 20 disk configurations.

Table 7.1 lists the XOR-based codes analyzed by the redundancy placement algorithms. The Hamming distance and MEV is listed for each code. The MEL is used to calculate the RME and so is more useful than the MEV for understanding the results in this section. The MEL for the (4,4)-RAID 10 is given in Section 7.1.1 and (5,3)-FLAT is given in Section 3.4.2. The MEL of the (6,2)-FLAT is $\{(s_0, s_1)$, (s_2, s_3), (s_2, s_6), (s_3, s_6), (s_4, s_5), (s_4, s_7), $(s_5, s_7)\}$. The MEL for the larger codes is too verbose to list. The FTV

code	Hamming dist. (d)	MEV	parity bitmaps
(6,2)-FLAT	2	$(0, 7)$	$15, 51$
(5,3)-FLAT	2	$(0, 1, 10)$	$7, 11, 29$
(4,4)-RAID 10	2	$(0, 4, 0, 0)$	$1, 2, 4, 8$
(10,2)-FLAT	2	$(0, 18)$	$127, 911$
(9,3)-FLAT	2	$(0, 5, 34)$	$31, 227, 365$
(17,3)-FLAT	2	$(0, 19, 162)$	$1023, 31775, 105699$
(16,4)-FLAT	2	$(0, 5, 80, 315)$	$511, 7711, 26215, 43691$

Table 7.1: XOR-based erasure codes.

is used for comparison purposes because the reliability simulated based on on the FTV approximates the median reliability over all possible placements.

The specific XOR-based codes listed in Table 7.1 were selected because, for the given values of k and m, they are the most fault tolerant XOR-based codes [65]. The only exception to this selection process is the (4,4)-RAID 10 which was selected because it has a familiar structure. The specific values of k and m were selected because the best codes have a Hamming distance of 2. It takes many CPU days for the HFR Simulator to simulate a single data loss event for more fault-tolerant codes, and so we restricted the Hamming distance to ensure that the results of the redundancy placement algorithms could be validated via simulation.

Beyond the XOR-based codes, some MDS codes are included in the evaluation to provide context. The placement of such codes does not affect their reliability because all sets of device failures of size d lead to data loss.

All of results presented in this section are MTTDL values measured in hours. The HFRS v.1 was used to produce all the MTTDL values. Except where noted, MTTDL values in tables and annotated on histograms are based on simulations of 1000 data loss events. The MTTDL values for data points in histograms are based on only 100 data loss events and so exhibit greater variance.

Table 7.2 lists MTTDL values for all the codes evaluated in this section based on homogeneous configurations. Two such configurations are listed: one based on 100k devices and the other based on 500k devices. Both device failure models only include disk

code	100k	500k
(6,2)-FLAT	3.99×10^7	9.66×10^8
(5,3)-FLAT	2.88×10^8	6.89×10^9
(4,4)-RAID 10	6.59×10^7	1.83×10^9
(7,1)-MDS	1.01×10^7	2.55×10^8
(10,2)-FLAT	1.54×10^7	3.89×10^8
(9,3)-FLAT	5.28×10^7	1.40×10^9
(11,1)-MDS	4.06×10^6	1.03×10^8
(17,3)-FLAT	1.44×10^7	3.55×10^8
(16,4)-FLAT	5.42×10^7	1.32×10^9
(19,1)-MDS	1.55×10^6	3.60×10^7

Table 7.2: MTTDL Homogeneous config.

failure and recovery; sector failure and scrubbing is not included in these simulations. Obviously, 500k homogeneous configurations are more reliable than 100k homogeneous configurations.

7.2.1 Eight Disk Configurations

In this section we exhaustively evaluate the three XOR-based codes of size 8 on various configurations. First, consider the 4-bimodal distribution. Figures 7.1, 7.2, and 7.3 respectively show MTTDL histograms for (4,4)-RAID 10, (5,3)-FLAT, (6,2)-FLAT. These histograms are constructed by simulating the MTTDL of the 8! = 40320 distinct placements. The simulations are based on devices that exhibit only disk failures and recoveries, not latent sector failures.

Each histogram is annotated with a vertical line. The vertical line corresponds to the MTTDL for the FTV. The FTV is described in Section 5.3.2 and, as discussed below, it estimates the MTTDL of the median placement. In these figures, the FTVMTTDL is indeed near the median MTTDL.

Each histogram is also annotated with a series of lines labeled with integers. These lines are related to RME calculations. For each of these codes, the BF-RP algorithm was used to determine the RME of each distinct placement. We were surprised to discover that for each of these codes, only a small number of distinct RME values were

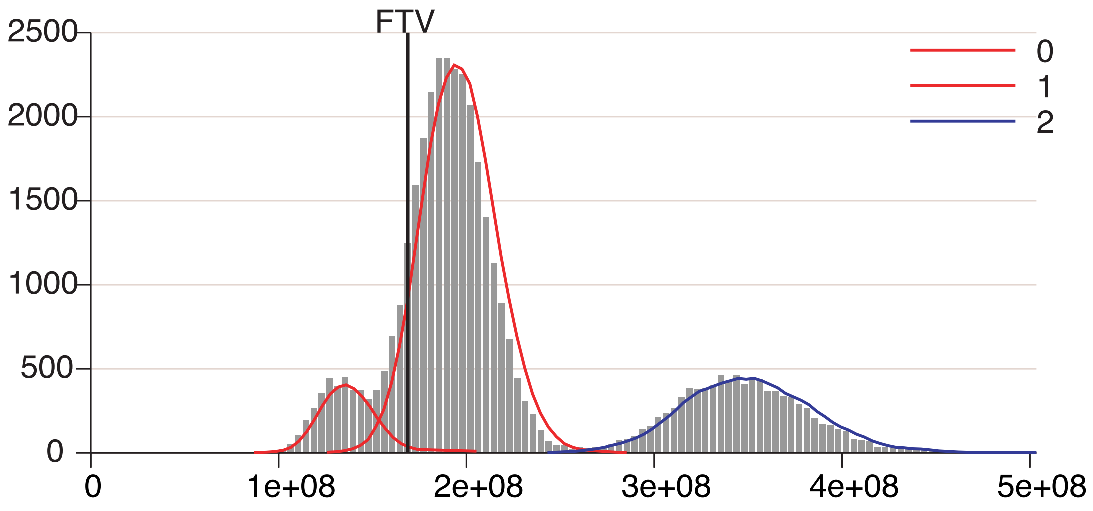


Figure 7.1: (4,4)-RAID10, 4-bimodal.

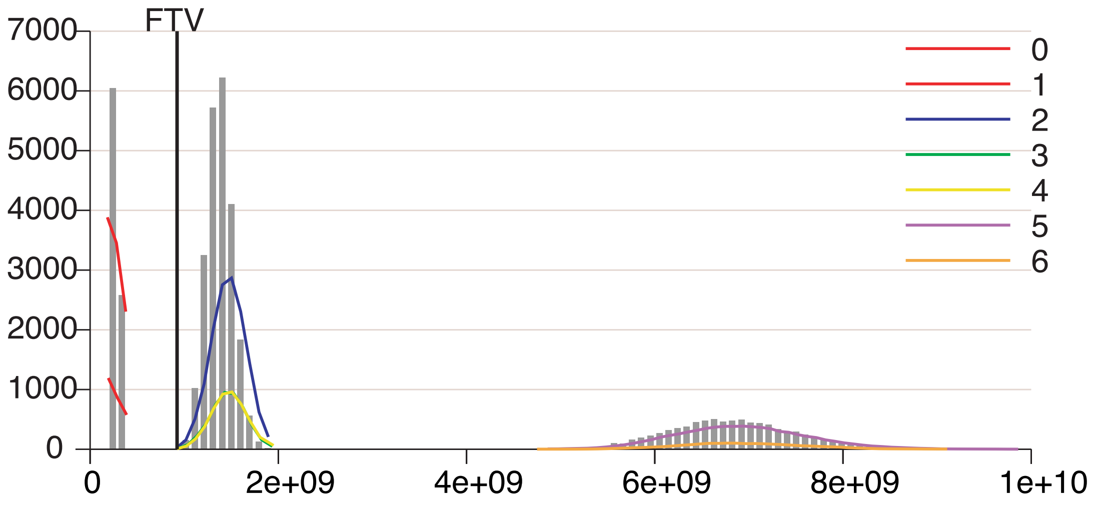


Figure 7.2: (5,3)-FLAT, 4-bimodal.

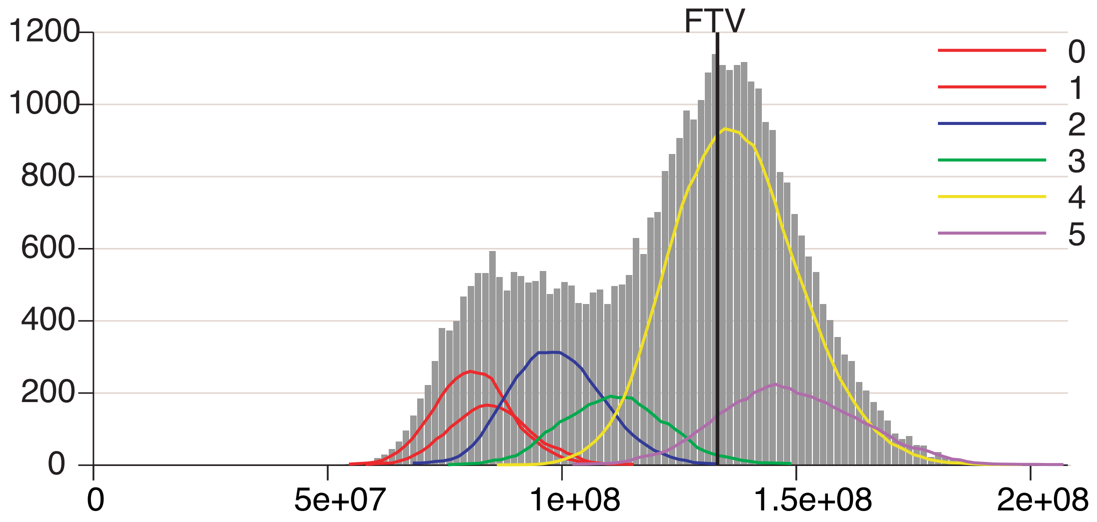


Figure 7.3: (6,2)-FLAT, 4-bimodal.

produced. From this, we hypothesized that there are *isomorphic placements*, i.e., different placements that have the same MTTDL. Each line on each histogram is effectively a sub-histogram for an isomorphic class of placements. The integer labels on the classes are in order of RME value, so line 0 has a lower RME than line 1. These sub-histograms strongly support our hypothesis that the RME correctly orders different placements with regard to reliability.

To better understand isomorphic placements, consider (4,4)-RAID 10. The following are example placements for each isomorphic placement class: 0 : (s_1, s_3, s_5, s_7, s_0, s_2, s_4, s_6), 1 : (s_0, s_1, s_2, s_4, s_3, s_5, s_6, s_7), and 2 : (s_0, s_1, s_2, s_3, s_4, s_5, s_6, s_7). The first four symbols in each placement is on a 100k device, and the second four symbols are on a 500k device. We already discussed the placements for classes 0 and 2 in Section 7.1.1. The placement for class 1 is consistent with the prior discussion: one pair of replicated symbols is on the 100k devices and so we expect the MTTDL to fall between class 0 (two pairs of replicated symbols on 100k devices) and class 2 (no pairs of replicated symbols on 100k devices).

The MEL for (5,3)-FLAT and (6,2)-FLAT is less regular than that of (4,4)-RAID 10, and so there are more isomorphic placement classes. The distribution of isomorphic placement classes is interesting: there tends to be many fewer placements in

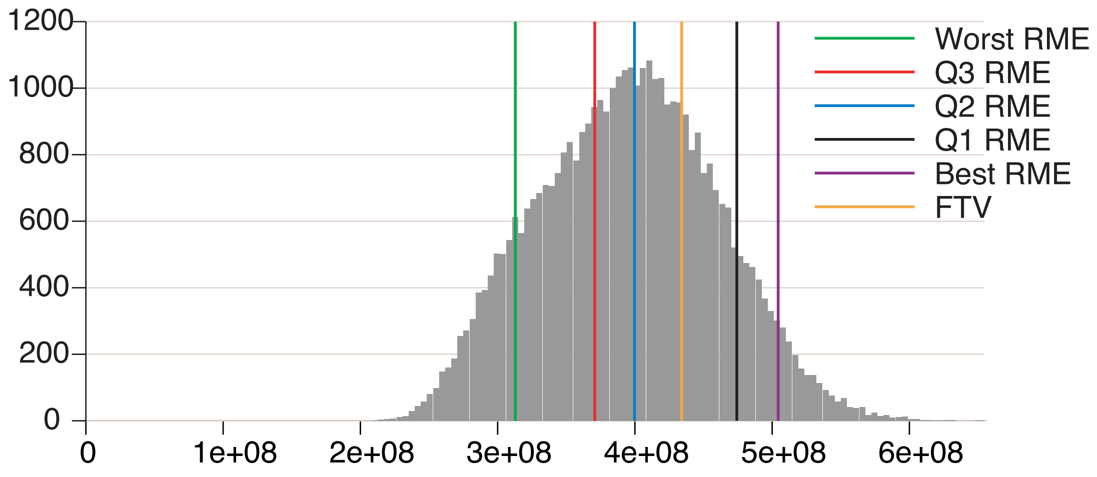


Figure 7.4: (4,4)-RAID10, uniform.

the best classes than in median classes. This suggests that good placement are less common.

Now consider the uniform configuration instead of the 4-bimodal configuration. Figures 7.4, 7.5, and 7.6 respectively show MTTDL histograms for (4,4)-RAID 10, (5,3)-FLAT, (6,2)-FLAT. These histograms are constructed by simulating the MTTDL of each of the $8! = 40320$ placements for the code on devices that exhibit only disk failures and recoveries. The one exception being Figure 7.5, that includes sector failures in the device model The BF-RP algorithm was used to determine the RME for every placement simulated.

The FTVMTTDL is annotated on these histograms. Sub-histograms for isomorphic placement classes are not presented. The uniform configuration leads to too many such classes to illustrate (105, 840, and 280 respectively). Instead, a vertical line is shown for a placement from each of the following isomorphic placement classes: Worst RME, Q3 (third quartile) RME, Q2 (second quartile) RME, Q1 (first quartile) RME, and Best RME.

Our hypothesis was that the MTTDL of the placement from Worst RME class would be less than that of the placement from the Q3 RME class, and so on. The results mostly support this hypothesis. There are two exceptions: results for Q1 RME and Q2 RME for (5,3)-FLAT are out of order, as are Best RME and Q1 RME for (6,2)-FLAT. We

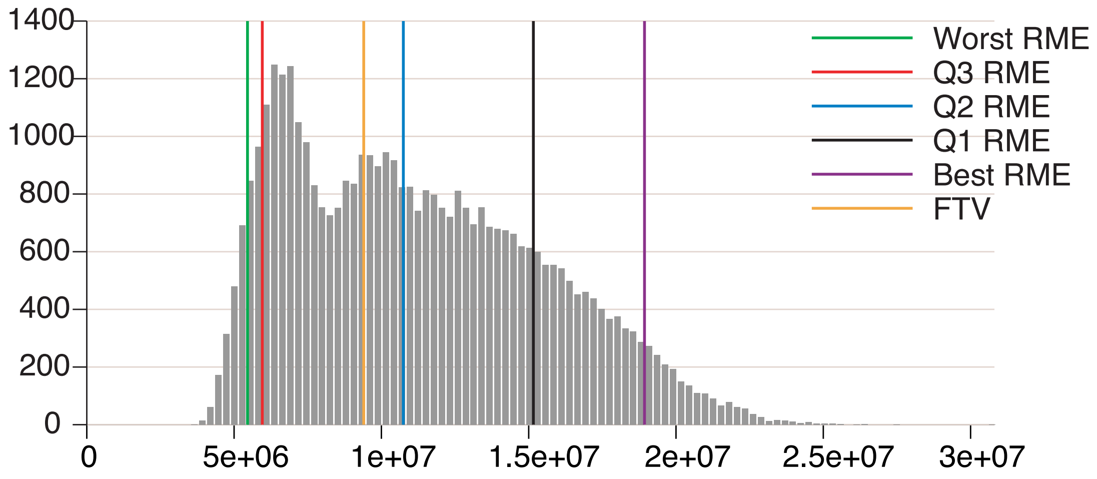


Figure 7.5: (5,3)-FLAT, uniform, sector.

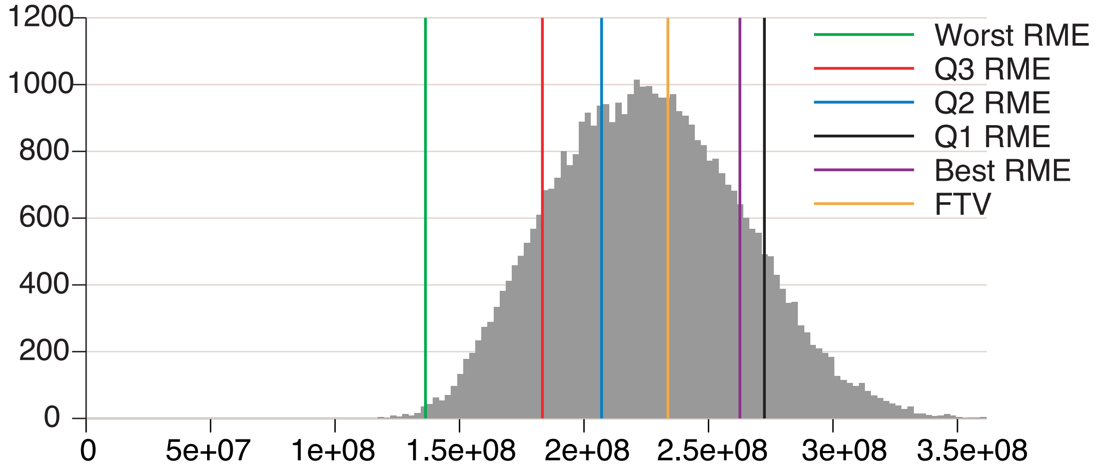


Figure 7.6: (6,2)-FLAT, uniform.

have looked at the specific placements in detail, as well as their RME values, and, based on back-of-the-envelope calculations, believe that the BF-RP has correctly ordered these placements.

We hypothesized that the FTVMTTDL, because it is based on a probability vector derived from the MEV, would provide a rough estimate of the median placement. We therefore expected the FTVMTTDL to align closely with the Q2 RMEMTTDL. These histograms support this hypothesis, and so we use the FTVMTTDL as a reference with which to compare the MTTDL of other placements.

Table 7.3 summarizes results for all of the bimodal configurations and the uniform configuration. For each code, the FTVMTTDL and the Best RMEMTTDL are listed. In all cases, the Best RMEMTTDL is better than that of the FTVMTTDL.

7.2.2 Sector failures

When we developed the RME metric, we assumed that sector failures would have a secondary effect on placement decisions and so could be excluded from the RME metric. This assumption appears to be valid, however some disclaimers are warranted. Figures 7.7 and 7.8 are based on the same 4-bimodal configuration setup respectively used for Figures 7.2 and 7.3, but use device failure models that include sector failure and scrubbing.

The histograms for (5,3)-FLAT met our expectations. Sector failures cause the MTTDL for the various placements to reduce significantly, but the shape of the histogram is preserved: redundancy placement affects MTTDL. The histograms for (6,2)-FLAT surprised us: sector failures in the device model result in redundancy placement having little effect on MTTDL. Studying the MEL for each code allows us to understand this result Both codes have Hamming distance 2, and thus some minimal erasures of size 2. In (5,3)-FLAT, only the symbols s_4 and s_7 occur in a minimal erasure of size 2, while *all* of the symbols in (6,2)-FLAT occur in such a minimal erasure. Reviewing the simulation results for (6,2)-FLAT, we note that the cause of data loss is single disk-single sector failures. In this specific scenario, redundancy placement has little effect on MTTDL because all of the device failure models have the same sector failure rate.

	(4,4)-RAID 10		(5,3)-FLAT		(6,2)-FLAT		
config.	FTV	Best RME	FTV	Best RME	FTV	Best RME	(7,1)-MDS
1-bimodal	8.60×10^8	8.39×10^8	3.30×10^9	6.90×10^9	4.98×10^8	6.20×10^8	1.19×10^8
2-bimodal	4.74×10^8	5.97×10^8	1.94×10^9	6.53×10^9	2.94×10^8	3.74×10^8	6.71×10^7
3-bimodal	3.01×10^8	4.35×10^8	1.23×10^9	6.40×10^9	1.72×10^8	2.54×10^8	4.41×10^7
4-bimodal	1.69×10^8	3.49×10^8	9.24×10^8	6.37×10^9	1.33×10^8	1.47×10^8	2.96×10^7
5-bimodal	1.51×10^8	1.81×10^8	6.07×10^8	6.62×10^9	8.76×10^7	9.67×10^7	2.05×10^7
6-bimodal	1.09×10^8	1.19×10^8	4.53×10^8	6.89×10^9	6.50×10^7	7.42×10^7	1.61×10^7
7-bimodal	8.29×10^7	8.42×10^7	3.40×10^8	1.35×10^9	4.99×10^7	5.18×10^7	1.26×10^7
uniform	4.34×10^8	4.88×10^8	1.56×10^9	6.11×10^9	2.34×10^8	2.79×10^8	5.60×10^7

Table 7.3: MTTDL of 8 disk configurations.

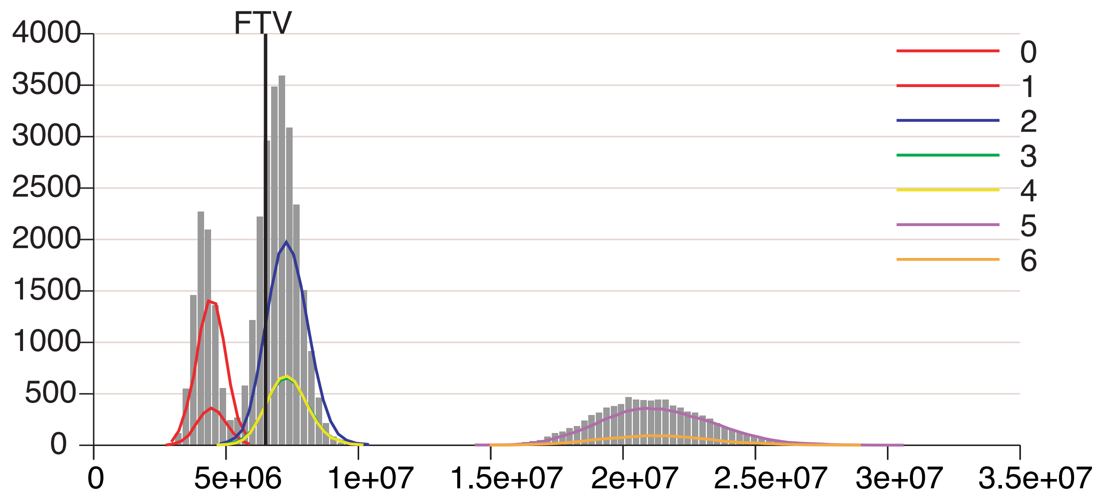


Figure 7.7: (5,3)-FLAT, 4-bimodal, sector.

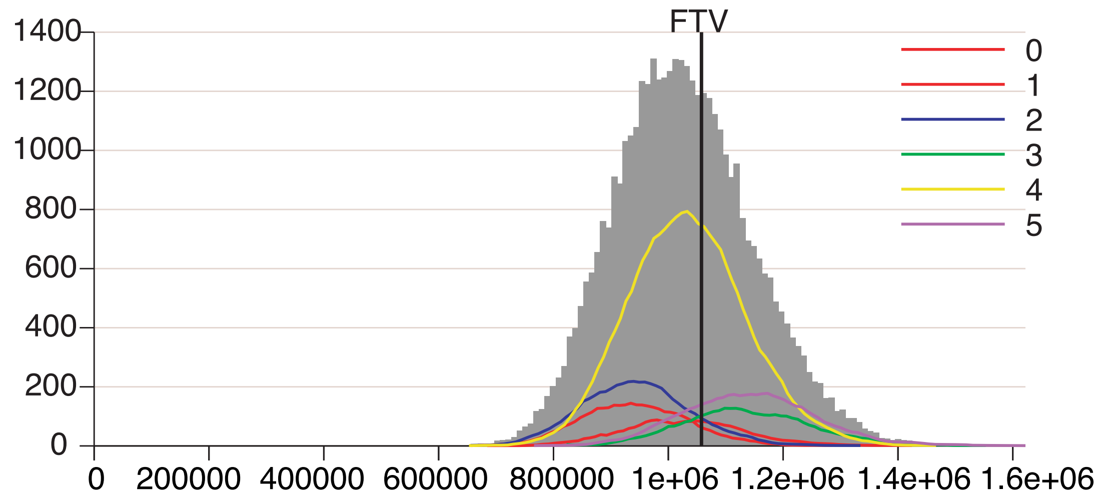


Figure 7.8: (6,2)-FLAT, 4-bimodal, sector.

	(9,3)-FLAT		(10,2)-FLAT		
configuration	FTV	Best RME	FTV	Best RME	(11,1)-MDS
3-bimodal	3.45×10^8	7.88×10^8	9.83×10^7	1.31×10^8	3.13×10^7
6-bimodal	1.57×10^8	3.30×10^8	4.25×10^7	5.14×10^7	1.27×10^7
9-bimodal	8.81×10^7	1.06×10^8	2.57×10^7	2.54×10^7	5.75×10^6
uniform	2.70×10^8	5.63×10^8	8.01×10^7	9.59×10^7	2.77×10^7

Table 7.4: MTTDL of 12 disk configurations.

7.2.3 Twelve Disk Configurations

For 12 disk configurations, it is not feasible to evaluate every possible placement via simulation, but it is feasible to do so via the RME metric. We ran the BF-RP algorithm for the (9,3)-FLAT, and (10,2)-FLAT codes for all possible bimodal configurations and the uniform configuration. We also ran the SA-RP algorithm on these configurations. In all cases, the SA-RP algorithm identified a placement from the same isomorphic placement class as the BF-RP algorithm (i.e., its RME was the same as the Best RME).

To determine the quality of the placements selected by the BF-RP and SA-RP algorithms, we simulated the Best RMEMTTDL and the FTVMTTDL for a subset of configurations. The results are listed in Table 7.4. In most cases, the MTTDL of the placement with the Best RME is significantly better than that of the FTV. For the 9-bimodal configuration, the MTTDL values for (10,2)-FLAT are effectively the same.

From the BF-RP results, we also can identify the Worst RME, Q3 RME, Q2 RME, and Q3 RME placements. We simulated the MTTDL of these placements as well as 1000 random placements to generate low fidelity histograms. Examples of such histograms for 6-bimodal configurations are given in Figures 7.9 and 7.10 for (9,3)-FLAT and (10,2)-FLAT respectively. These results further support our hypothesis that the RME metric correctly orders placements by reliability.

7.2.4 Twenty Disk Configurations

For 20 disk configurations, it is infeasible to evaluate every possible placement via simulation or the RME metric. Instead, we use the SA-RP algorithm to identify an

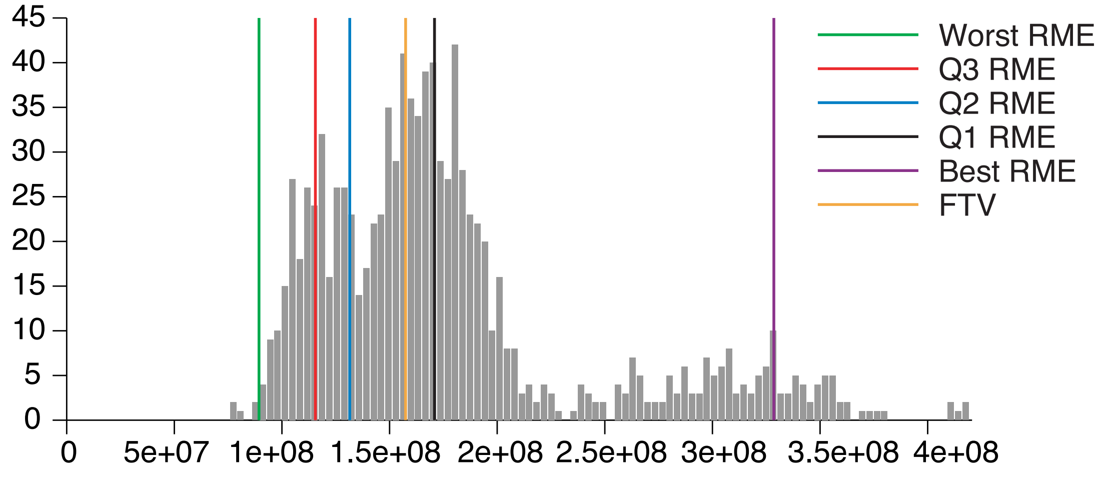


Figure 7.9: (9,3)-FLAT, 6-bimodal.

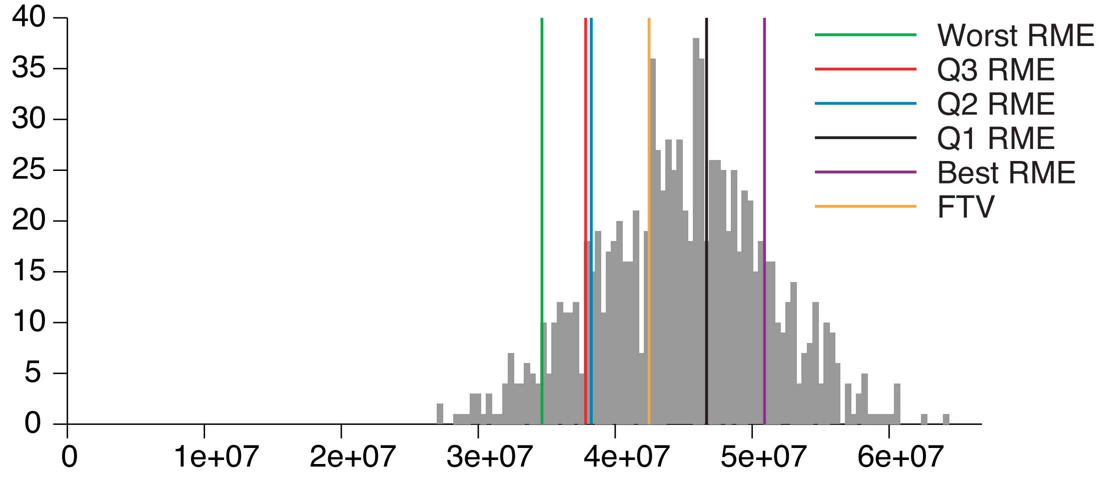


Figure 7.10: (10,2)-FLAT, 6-bimodal.

Approximate Best RME placement for these configurations. We ran the SA-RP algorithm for the (17,3)-FLAT, and (16,4)-FLAT codes for all of the bimodal configurations and the uniform configuration. We run the SA-RP algorithm for a total of 1000000 steps; if the RME does not improve in 25 steps, the placement reverts to the last best placement for this execution; if the best RME placement does not improve in 1000 steps, a new execution is initialized with a random placement.

To determine the quality of the placements selected by the SA-RP algorithm, we simulated MTTDL of the Approximate Best RME placement found by SA-RP and compare it with the FTVMTTDL for a subset of configurations. The results are listed in Table 7.5. In all cases, the Approximate Best RMEMTTDL is significantly better than the FTVMTTDL.

7.3 Conclusion

In this work, we have introduced the *redundancy placement problem* in which a mapping of the symbols in a flat XOR-based code onto a set of heterogeneous storage devices with known failure and recovery rates is found in a way that maximizes reliability. We solved this problem be developing a metric, called the Reliability MTTDL Estimate (RME), a simple model based on estimated device unavailability and the Minimal Erasures List (MEL), a compact description of the fault tolerance of an erasure code. Two redundancy placement algorithms were developed in an effort to find a placement that maximizes reliability. The first algorithm BF-RP is only feasible for small codes and find a highly reliable placement via brute-force search. The second algorithm, which is best suited for larger codes, uses simulated annealing to find a near-optimal placement with respect to reliability, called SA-RP.

We extended our work in Section 5.3 to provide an extensive empirical evaluation, which shows that the RME correctly orders different placements for a given code by MTTDL. Additional results suggest that placements found by SA-RP are significantly more reliable than the *median* placement. Finally, results of the BF-RP algorithm lead us to the existence of so-called *isomorphic placements.*

configuration	(17,3)-FLAT FTV	(17,3)-FLAT Best RME	(16,4)-FLAT FTV	(16,4)-FLAT Best RME	(19,1)-MDS	(18,2)-MDS
5-bimodal	8.94×10^7	1.29×10^8	3.59×10^8	1.39×10^9	9.63×10^6	1.50×10^{10}
10-bimodal	4.36×10^7	5.02×10^7	1.89×10^8	1.33×10^9	4.48×10^6	5.71×10^9
15-bimodal	2.33×10^7	2.61×10^7	8.72×10^7	2.85×10^8	2.53×10^6	1.94×10^9
uniform	8.49×10^7	9.62×10^7	3.38×10^8	8.71×10^8	8.49×10^6	1.27×10^{10}

Table 7.5: MTTDL of 20 disk configurations.

Chapter 8

Trading Reliability and Power-Efficiency using Power-Aware Coding

Traditionally, storage systems are measured in terms of performance and reliability. Due to the increasing amount of data stored in recent years and the significant amount of power required to store such data, a great deal of work has gone into measuring and minimizing the power consumption of storage systems [9, 61, 72, 48, 71, 60]. Energy has moved to the forefront of data center design as companies try to simultaneously reduce costs, maximize data center density, and push towards greener business practices. Storage accounts for roughly 27% of a data center's power budget [5]; thus, pro-actively activating and deactivating disks can effectively lower the energy footprint of a data center.

Whether data is mirrored or transformed via matrix operations, almost every large-scale storage system relies on erasure codes for reliability [54, 58, 57, 33, 1]. Most systems assume the existence of a k-of-n encoding scheme to protect data without considering the underlying structure of the code. We find that two distinct k-of-n codes may have very different performance and reliability properties that we can exploit to intelligently partially reconstruct data.

In this chapter we present a technique called *power-aware coding*. We find that in addition to fault-tolerance and performance, the structure of an erasure code may be exploited to save power in a storage system. As an example, consider a simple RAID4 disk array with no failures. Since the array can tolerate any one disk failure, any single

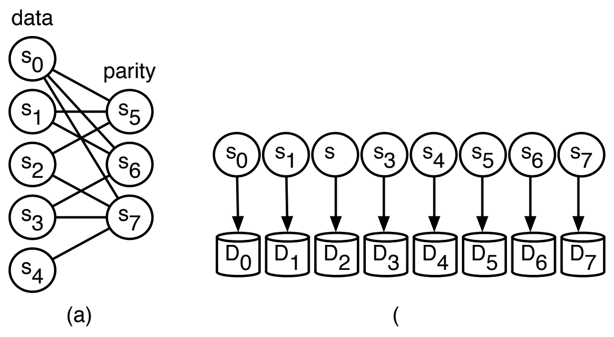


Figure 8.1: Code instance of a (5,3)-FLAT code across 8 devices.

disk in the array can be deactivated for an extended period of time and need not be activated during a read request. Instead of activating the device, its contents can be reconstructed from the remaining devices.

We define *power-aware coding* in terms of a set of devices and an erasure code instance across the devices. Assume an erasure code contains a total of n symbols and a system of N devices. A *code instance* is a mapping of code symbols to devices. An example code instance is shown in Figure 8.1-b. There are $N = n$ devices and a one-to-one mapping of code symbols to devices. Each code symbol, s_i, is mapped to disk D_i.

The crux of power-aware coding is to prevent spinning up inactive disks when servicing read requests by treating each inactive disk as an erasure. As an example, consider the setup shown in Figure 8.1. Suppose disks D_0, D_5, D_6 and D_7 are active and all others are inactive. If the system receives a read request for disk D_4, we can service the request as $D_0 \oplus D_5 \oplus D_6 \oplus D_7$ instead of activating disk D_4, since $s_4 = s_0 \oplus s_5 \oplus s_6 \oplus s_7$.

This chapter provides both theoretical and applied contributions to the study of power-aware, erasure-coded systems. This chapter provides seven contributions: four theoretical contributions and three applied contributions.

The theoretical portion of this work is rooted in a measure we call *reconstructability.* First, we describe reconstructability, which is a generalization of traditional fault tolerance and applies to both MDS and non-MDS codes (Section 8.1). Next, reconstructability is used to describe a property called the *balanced property*

(Section 8.2). An available set of symbols, called an ASSET (Available Symbol SET), is balanced if the number of symbols in the ASSET equal to the number of solvable data symbols. Third, we describe how to generate balanced ASSETs. Given a code, a simple algorithm is used to generate all possible balanced ASSETs. Additionally, balanced ASSETs can be generated, allowing a designer to build codes around a set of balanced ASSETs. Fourth, we define a system-level requirement, called *immediate parity update*, and the *single-data connected-parity* (SDCP) policy, which enforces this requirement (Section 8.3). An immediate parity update system ensures that all parity information is updated immediately when writing user data. A bound is placed on the rate of a code that has balanced ASSETs in a SDCP enforced system.

The applied portion of this chapter describes the architecture of an immediate parity update system. First, we describe a power-managed, log-structured system that uses device spin-down to save power (Section 8.4). The system utilizes a SDCP-like policy to absorb a write workload and specialized reconstruction algorithms to recover data on inactive devices. Next, we derive three metrics for use in a power-managed system: aggregate write group size, aggregate reconstructability, and aggregate relative gain. The proposed metrics are used to trade power, reliability and space efficiency in a power-aware storage system (Section 8.5). We have selected 17 MDS and non-MDS codes to perform the tradeoff analysis.

8.1 Reconstruction of Inactive, Erasure-Coded Symbols

Erasure codes are typically used to provide fault tolerance in storage systems and communication channels. In general, k data symbols are transformed into $n = k+m$ code symbols in a way that tolerates 1 to m symbol erasures. Such an erasure code is called *systematic* due to the separation of data and parity symbols. From now on, when we say erasure code we mean systematic erasure code. Recall that an erasure is a *known* error. That is, a symbol is erased if it is known to be in error or the information corresponding to the symbol is lost. Examples of erased symbols include failed storage devices, unreadable sectors and dropped packets. The use of erasure codes enables the recovery of failed storage devices, unreadable sectors and dropped packets.

In addition to providing fault tolerance, erasure codes can also be used to recover symbols on inactive storage devices. Consider traditional fault tolerance in an erasure coded codeword

Definition 8.1.1. *An n-symbol codeword can* ***tolerate*** *an f-symbol erasure pattern if the $n - f$ available symbols can be used to recover the original k data symbols.*

Assume that the code symbols are mapped to storage devices that are either energized (active) or powered off (inactive). Define all symbols mapped to inactive devices as erased and the symbols mapped to active devices as the *Available Symbol Set* (ASSET). Suppose there are f symbols mapped to inactive devices. By Definition 8.1.1, if the $n - f$ available symbols can recover the k data symbols, then all read requests to inactive data symbols may be served from the available symbols without activating any additional devices.

The traditional definition of fault tolerance can be relaxed to allow *partial reconstruction* of the data symbols.

Definition 8.1.2. *An n-symbol codeword can* ***partially tolerate*** *(reconstruct) an f-symbol erasure pattern if the $n - f$ available symbols can be used to recover more than* 1 *and no more than $k - 1$ of the original data symbols.*

By Definition 8.1.2, the $n - f$ available symbols may, in some cases, be used to avoid device activation when attempting to access a symbol on an inactive device. The key to exploiting partial reconstruction is to maximize the reconstruction potential of inactive symbols.

Whether fault-tolerance is presented in terms of full or partial reconstruction, the reconstruction capability of available code symbols is fully captured by a metric called *reconstructability.*

Definition 8.1.3. ***Reconstructability****, $R(C, s)$, is the number of directly readable or solvable data symbols given an erasure code, C, and* ASSET*, s.*

Suppose s contains t symbols, k' of which are data symbols. By default, the reconstructability of s must be at least k', regardless of the underlying erasure code. By Definition 8.1.3, $R(C, s) - k'$ inactive symbols can be reconstructed from the t active symbols in s.

In general, reconstructability is mostly dependent on the structure and rate of the underlying erasure code. There are two distinct structural classes of erasure codes: MDS and non-MDS . The difference between MDS and non-MDS codes is illustrated by exploring the fault-tolerance and parity equations of each class.

Recall that the *Hamming distance* of an erasure code provides a compact description of a code's fault tolerance. Any set of symbol erasures strictly less than the Hamming distance of a code can be tolerated. An MDS erasure code with k data symbols and m parity symbols, denoted (k,m)-MDS, has a Hamming distance of $m + 1$. Therefore, a (k,m)-MDS code can tolerate up to any m symbol erasures out of n total symbols. A non-MDS erasure code with k data symbols and m parity symbols, denoted (k,m)-NON-MDS, has a Hamming distance strictly less than $m + 1$. This means that a (k,m)-NON-MDS code can tolerate many, but not all, erasure patterns of size at most m.

Each of the m parity symbols in an erasure code can be expressed as a linear combination of the k data symbols. For a MDS code, the coefficients of each linear combination must be non-zero, otherwise the code will not be able to recover the k data symbols in at least one erasure pattern containing less than m symbols. As a result, every entry in each parity column of the systematic generator matrix of an MDS code must be non-zero. Consider the following systematic generator matrix for an (4,2)-MDS code with symbols s_0, s_1, s_2, s_3, s_4 and s_5

$$\begin{array}{c} \begin{array}{cccccc} s_0 & s_1 & s_2 & s_3 & s_4 & s_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & \alpha_0 & \beta_0 \\ 0 & 1 & 0 & 0 & \alpha_1 & \beta_1 \\ 0 & 0 & 1 & 0 & \alpha_2 & \beta_2 \\ 0 & 0 & 0 & 1 & \alpha_3 & \beta_3 \end{pmatrix} \end{array}$$

where the α_i and β_i are elements of a finite field. Now suppose $\alpha_1 = 0$. An (4,2)-MDS code can tolerate the loss of any 2 symbols, but when $\alpha_1 = 0$, the code cannot tolerate the loss of symbols s_5 and s_1. In other words, zeroing out the columns associated with symbols s_5 and s_1 will result in a matrix with rank 3. Since there are 2 parity symbols and there exists at least one intolerant erasure pattern of size 2, the code is not MDS .

In general, a zero entry in the parity submatrix of any systematic (k,m)-CODE

code will result in a code with a Hamming distance of at most m. It follows that the parity submatrix of a MDS code contains all non-zero entries. As a consequence, each parity column represents an equation with k unknown variables; each data symbol represents an unknown variable.

This insight leads to a fundamental property of systematic MDS erasure codes: if less than k symbols are available, no erased symbols can be recovered. Suppose there are k' available data symbols and m' available parity symbols. Each of the available parity symbols is an equation of the same $(k - k')$ unknown variables. If $(k - k') \leq m'$, then we can solve for the $(k - k')$ unknown variables (data symbols) using elementary linear algebra. However, when $(k - k') > m'$ the system of equations generated by the m' equations is under-determined and we cannot solve for the $(k - k')$ unknowns. In short, erased data symbols cannot be recovered when $k' + m' < k$. A similar argument applies to erased parity symbols: k available or recovered data symbols are needed to recover an erased parity symbol. The formal description of these observations is stated in Property 8.1.4.

Property 8.1.4. *At least k symbols must be available in order to recover any erased symbol in a systematic (k,m)-*MDS *code.*

The parity columns associated with a non-MDS code may have entries with a value of zero. While such codes will not provide optimal fault-tolerance for parameters k and m, recovery of an erased symbol may involve less than k available symbols. Consider the following systematic generator matrix for a (4,2)-NON-MDS code

$$\begin{array}{c} \begin{array}{cccccc} s_0 & s_1 & s_2 & s_3 & s_4 & s_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Suppose symbol s_3 is currently inactive. In order to reconstruct the data associated with s_3 only symbols s_2 and s_5 need to be active.

Property 8.1.5. *In many cases, less than k available symbols are required to recover a single erased symbol in a systematic (k,m)-*NON-MDS *code. This follows from the fact* *that parity columns in the systematic generator matrix for a (k,m)*-NON-MDS *code may have zero entries.*

Properties 8.1.4 and 8.1.5 are quite substantial and highlight the key difference between the reconstructability of MDS and non-MDS codes. A (k,m)-NON-MDS code will typically require less available symbols than an (k,m)-MDS code when recovering an erased symbol. This turns out to be very important when exploiting redundancy in a power-managed system.

Here we combine Properties 8.1.4 and 8.1.4 with Definition 8.1.3 to place bounds on the reconstructability of MDS and non-MDS codes. Assume an erasure code with k data symbols and m parity symbols. Additionally, let s be the ASSET that contains k' data symbols and t total symbols. The reconstructability for any MDS code (C) under these parameters is

$$R(C, s) = \begin{cases} k' & \text{, when } t < k \\ k & \text{, when } t \geq k \end{cases} . \tag{8.1}$$

This result is quite intuitive. Simply put, if the total number of available symbols is less than the number of data symbols, then the reconstructability of *any* systematic MDS code is equal to the number of available data symbols. If the number of available symbols is equal to or greater than the number of data symbols, then all data symbols can be reconstructed.

Next, consider a non-MDS code with the same parameters. While the reconstructability of an MDS code will be one of two values, a non-MDS code can have a much wider range. The reconstructability for any non-MDS code (C) under these parameters is

$$k' \leq R(C, s) \leq t \leq k. \tag{8.2}$$

These results show a clear distinction between MDS and non-MDS codes with respect to reconstructability. Assume t symbols are in an ASSET. When $t < k$, the reconstructability of an MDS code is maximized only when all t symbols in the ASSET are data symbols. Once t exceeds k, the reconstructability will be k regardless of the symbols in the ASSET. For non-MDS codes, it turns out that the reconstructability can be maximized when $t < k$ and the ASSET is a mixture of data and parity symbols.

Equations 8.1 and 8.2 can be combined to define the maximum reconstructability of any erasure code.

Definition 8.1.6. *The **maximum reconstructability** of a (k,m)-CODE code with t active symbols is k when $t > k$ and t otherwise.*

Definition 8.1.6 describes the optimal reconstructability of any ASSET given a (k,m)-CODE code. If the number of available symbols is equal to maximum reconstructability, then we call the symbol set *balanced.* In other words, the number of active symbols equals the number of reconstructable data symbols. In terms of utility to a power-managed system, the existence of balanced ASSETs is interesting.

As an example, suppose t data and parity symbols are currently active such that all dependent parity for at least one active data symbol is active. User data can be written to the system and all dependent parity can be immediately updates. If the t available symbols represent a balanced ASSET, then t data symbols can be read without activating any additional devices. The balanced property enables parity updates to finish in a timely fashion, while maximizing the amount of user data that can be read without additional activations.

Property 8.1.7. *If the maximum reconstructability of a (k,m)-CODE code with an ASSET of t symbols equals t, then the ASSET is a **Balanced Available Symbol Set** (BASSET) or a t-BASSET.*

A simple example of a BASSET is the set that contains all data symbols. In practice an ASSET will contain a mixture of data and parity symbols, implying that in certain cases non-MDS codes may be better suited to power-aware coding than MDS codes.

8.2 Structure of Balanced Available Symbol Sets

Section 8.1 introduced the idea of reconstructability for erasure codes and an ASSET. We find that the maximum reconstructability for any erasure code equals the size of the ASSET or k if the size of the ASSET is greater than k. Any ASSET of size k

or less that achieves this optimum for an erasure code is balanced. In this section we explore how both erasure code and ASSET choice lead to this balanced property.

We focus on XOR-based, non-MDS codes, denoted NXOR or (k,m)-NXOR. In the rest of this chapter, we consider the terms non-MDS and XOR-based non-MDS to be synonymous. While much of our discussion thus far applies to any non-MDS code, we use XOR-based codes to simplify the analysis. This simplification is due to the size of the finite field used to construct the generator matrix. In general, the elements of the generator matrix of a non-MDS code is built using the Galois field of 2^l elements. The generator matrix of an XOR-based code is based on the field of 2 elements: 0 and 1. Much of our analysis is built on top of elementary matrix operations on a code's generator matrix. Obtaining crisp results is much more straightforward when all elementary operations are performed on columns with entries from $GF(2)$.

An ASSET (or BASSET) with t symbols for a (k,m)-CODE code is represented by a $k \times t$ matrix. The matrix is induced by the corresponding columns from a code's generator matrix. The elements of the matrix will consist of elements from $GF(2)$ for an NXOR and $GF(2^l)$ for an MDS code with $m > 1$.

In general, the structure of a BASSET can be determined using elementary linear algebra. That is, treating the columns of the generator matrix as $k \times 1$ vectors and performing elementary operations on these vectors. The generator matrix, G, of a code contains a $k \times k$ identity matrix formed by the first k columns and a $k \times m$ parity submatrix that contains columns that are linear combinations of the first k columns. The first k columns correspond to the data symbols of the code, while the last m columns correspond to the parity symbols. If the code has a total of n symbols, $s_1, s_2, \ldots, s_n$, then $s_i = e_i$, where $1 \leq i \leq k$ and e_i is the elementary column with a 1 in the i-th entry.

Finding a t-BASSET begins by choosing elementary columns that correspond to t different data symbols. The matrix induced by these columns can be transformed into another matrix via elementary column operations. A BASSET is found whenever the columns of the transformed matrix have corresponding columns in the generator matrix of the code.

Definition 8.2.1. *Let $c_{i_1}, c_{i_2}, \cdots, c_{i_l}$ be the column vectors associated with code symbols $s_{i_1}, s_{i_2}, \cdots, s_{i_l}$. We can get the reduced-column echelon form of $[c_{i_1}, c_{i_2}, \cdots, c_{i_l}]$,*

M, by applying elementary column operations. The number of unit column vectors in M equals the number of solvable data symbols. It is well-known that each elementary column operation can be reversed in a way that yields $[c_{i_1}, c_{i_2}, \cdots, c_{i_l}]$ from M, thus every $c_{i_1}, c_{i_2}, \cdots, c_{i_l}$ is a linear combination of the columns of M.

Definition 8.2.1 can be used to place a restriction on the structure of the symbols in a BASSET.

Theorem 8.2.2. *Suppose an* ASSET *containing code symbols $s_{i_1}, s_{i_2}, \ldots, s_{i_t}$ is a t-*BASSET*. The matrix induced by the corresponding columns $c_{i_1}, c_{i_2}, \ldots, c_{i_t}$ must have $(k-t)$ all zero rows.*

Proof. This follows from Definition 8.2.1. Since the ASSET is a t-BASSET, $M = [c_{i_1}, c_{i_2}, \cdots, c_{i_t}]$ can be reduced to $M' = [e_{i_1}, e_{i_2}, \cdots e_{i_t}]$ via elementary column operations. It follows that $e_{i_j} \neq e_{i_k}$, where $j \neq k$, otherwise the ASSET would not be t-balanced. Thus, M' has $(k-t)$ all-zero rows and by Definition 8.2.1 so must M. □

Definition 8.2.1 and Theorem 8.2.2 provide guidelines for finding a BASSET given a set of symbols from an arbitrary erasure code. Building BASSETs using MDS and non-MDS codes is discussed in the remainder of this section.

8.2.1 BASSETs using MDS Codes

The regular structure of MDS codes simplifies reasoning about BASSETs. These codes have two types of BASSETs: the set containing all data symbols when $t < k$ and the set containing *any* combination of data or parity symbols when $t = k$. When $t > k$, the set cannot be balanced, since the number of active symbols exceeds the total number of data symbols in the underlying code.

Trivially, when using an MDS code and an ASSET of size $t = k$, all data symbols are solvable; thus, the set will always be balanced.

The other type of BASSET using MDS codes can be explained using Theorem 8.2.2 and Property 8.1.4. Suppose the size of an BASSET is $t = k' + m'$, where the set contains k' data symbols and m' parity symbols. Property 8.1.4 states that the parity columns of an MDS code *must* consist of all non-zero entries. If $t < k$, then by Theorem 8.2.2, $(k-t) > 0$ rows of the matrix induced by the t symbols must be all zero. As a consequence, when $t < k$, a BASSET cannot contain parity symbols from an MDS code. This is true because there cannot be zero entries in the parity submatrix of an MDS code. Therefore, any BASSET built from a MDS code must contain *only* data symbols when $t < k$.

8.2.2 BASSETs using Non-MDS XOR-based Codes

NXOR codes are more complicated than MDS codes. As with MDS codes, an ASSET containing all data symbols will be balanced when $t \leq k$. In addition, when $t > k$, there is no way to make a BASSET using a NXOR code. Unlike MDS codes, a BASSET containing both data and parity symbols can be obtained when $t < k$. The existence of a BASSET or an ensemble of BASSETs depends solely on the underlying erasure code.

There are three ways to construct t-BASSETs based on a NXOR code. In the first construction, all $\binom{n}{t}$ possible ASSETs can be evaluated from a candidate (k,m)-NXOR code. In the second construction, all possible $\binom{2^k-1}{t}$ ASSETs can be evaluated from all $2^k - 1$ possible generator matrix columns for a (k,m)-NXOR code. The final construction can simply generate all possible t-BASSETs given the number of data symbols in the code. In the first construction, a set of BASSETs are built around a NXOR code. In the latter two constructions, a NXOR code can be built around a set of BASSETs.

8.2.2.1 Building t-BASSETs Around a Code

The brute-force algorithm used to find BASSETs, called BF-CODE-BASSET, relies on Definition 8.2.1 and Theorem 8.2.2. A $k \times t$ matrix is generated for each ASSET. If there are more or less than $(k - t)$ non-zero rows in the matrix, the ASSET is discarded; otherwise, the ASSET is added to a list of *candidate* BASSETs. Once all ASSETs have been evaluated using the row test, the matrices associated with the candidate BASSETs are placed in canonical form using elementary matrix operations. If the canonical form contains t unit columns, then the candidate BASSET is added to a list of BASSETs.

The basic algorithm is:

Input: a (k,m)-NXOR code and t

Step 1 Generate $\binom{n}{t}$ $k \times t$ binary matrices

Step 2 If a matrix has $(k-t)$ all-zero rows, add it to the candidate set S

Step 3 If the canonical form of a matrix in S has t elementary columns, then return it as a t-BASSET

Obviously, this algorithm requires $\binom{n}{t}$ ASSET evaluations. The BF-CODE-BASSET algorithm proceeds in two phases. In general, the first pass will require $\binom{n}{t} \cdot k \cdot n$ operations (check for $(k-t)$ rows) and the second pass will require $k \cdot n^2$ operations per candidate BASSET. The magnitude of the number of candidate BASSETs on the second pass is completely dependent on the underlying erasure code. As n increases, we expect the first pass check to assist in a lower running time.

8.2.2.2 Building Codes around t-BASSETs

There are two algorithms for building codes around t-BASSETs: BF-ALL-BASSET and GEN-ALL-BASSET. Both algorithms require t (BASSET size) and the number of data symbols in the code, k, as input. Similar to BF-CODE-BASSET, BF-ALL-BASSET performs a brute force search over all possible ASSETs with k data symbols and returns all t-BASSETs in the form of $k \times t$ binary matrices. The GEN-ALL-BASSET algorithm generates all t-BASSETs from $k \times t$ matrices containing only elementary columns.

The BF-ALL-BASSET algorithm requires a search over all possible $k \times t$ binary matrices. There are a total of $2^{k \cdot t}$ $k \times t$ binary matrices. In most cases, finding BASSETs in the space of $2^{k \cdot t}$ matrices is intractable. The space of matrices resulting in BASSETs can be drastically cut down using a few pieces of information. Every matrix induced by the symbols of a BASSET must have distinct columns. That is, no two columns can be the same. Additionally, every column must have at least one non-zero entry. These two pieces of information cut the search space down from $2^{k \cdot t}$ to $\prod_{i=1}^{t} \left(2^k - i\right)$.

We use Theorem 8.2.2 to drastically cut down the search space even further. By Theorem 8.2.2, the matrix corresponding to a BASSET will have $(k-t)$ all-zero rows. There are $\binom{k}{k-t}$ ways to choose $(k-t)$ all-zero rows. This means that the search space can be reduced to $\binom{k}{k-t} \cdot \prod_{i=1}^{t} \left(2^t - i\right)$ possible matrices. An algorithm similar to BF-CODE-BASSET can be used to find BASSETs over the $\binom{k}{k-t} \cdot \prod_{i=1}^{t} \left(2^t - i\right)$ candidate BASSETs.

An alternative to the BF-ALL-BASSET algorithm is the GEN-ALL-BASSET algorithm. Instead of performing a brute-force search over a set of candidate matrices, the GEN-ALL-BASSET algorithm generates every BASSET of size t for a NXOR code with k data symbols. The intuition behind the GEN-ALL-BASSET algorithm is based on Definition 8.2.1. The most basic BASSET of size t is the BASSET of all data symbols. By Definition 8.2.1, performing elementary column operations on a matrix induced by a BASSET will result in another BASSET.

The GEN-ALL-BASSET algorithm begins by enumerating all possible $k \times t$ matrices containing t distinct elementary columns. There are $\binom{k}{t}$ such matrices, called *elementary* BASSETs. Each elementary BASSET will generate 2^{t^2-t} BASSETs as follows. Each column can be recomputed as the sum of itself and any other column in the matrix. Each time a column is recomputed, a new BASSET is been generated. A column from an elementary BASSET can take on one of $\sum\limits_{i=0}^{t-1} \binom{t-1}{i}$ possible values. That is, each column can be the sum of itself and between 0 and $t-1$ other columns. As a result, each elementary BASSET generates

$$\left(\sum_{i=0}^{t-1} \binom{t-1}{i} \right)^t = 2^{t^2-t} \tag{8.3}$$

total BASSETs. Putting it all together, the GEN-ALL-BASSET algorithm will generate $\binom{k}{t} \cdot 2^{t^2-t}$ BASSETs. Note that the generated BASSETs are not necessarily unique; thus, whenever a duplicate BASSET is generated, it should be discarded.

8.2.3 Discussion

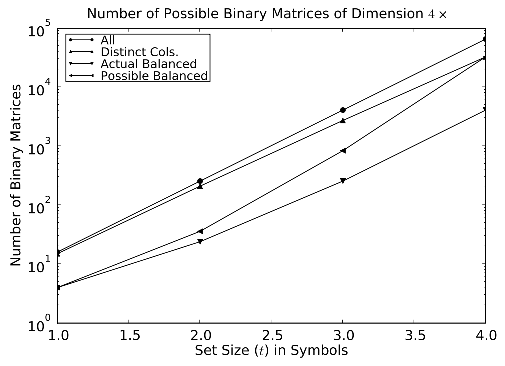
In this section we have shown the general structure of BASSETs and how they are created for both MDS and NXOR codes. In terms of a power-aware system, BASSETs provide a way to balance the total number of active symbols and number of available data symbols. In a way, a BASSET provides the accessibility of simply keeping only data symbols active, while allowing parity symbol updates to complete in a timely fashion.

There are two main takeaways from this section. First, reasoning about the reconstructability and existence of BASSETs is much easier for MDS codes than NXOR codes. Furthermore, NXOR codes provide much more flexibility when ASSETs have less than k symbols. NXOR codes will provide more opportunity for finding BASSETs among ASSETs containing a mixture of data and parity symbols. As we will explain in the subsequent sections, this can be beneficial to power-aware storage systems. Second, we have developed three algorithms for finding BASSETs in NXOR codes. Given an NXOR code, the BF-CODE-BASSET algorithm will find BASSETs via brute-force search. The BF-ALL-BASSET algorithm will find all t-BASSETs for a code with k data symbols using brute-force search. The GEN-ALL-BASSET algorithm generates all t-BASSETs for a code with k data symbols.

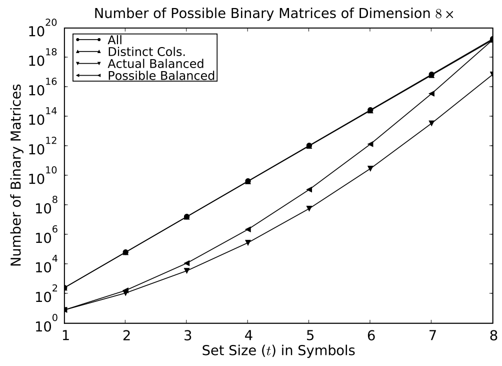
To get an idea of the space of BASSETs among all possible ASSETs, we briefly analyze the search space associated with BF-ALL-BASSET and GEN-ALL-BASSET. Figure 8.2 counts ASSETs for codes containing 4 (a), 8 (b), 16 (c) and 24 (d) data symbols. A line in each plot corresponds to all possible $k \times t$ matrices (All), all possible $k \times t$ matrices with distinct non-zero columns (Distinct Cols.), all possible $k \times t$ matrices with $(k - t)$ all-zero rows (Possible Balanced) and all $k \times t$ matrices that correspond to a BASSET (Actual Balanced).

There are two major pieces of information to take away from these figures. First, the BF-ALL-BASSET algorithm should only be used when the number of data symbols is small. Attempting to perform a brute force search for all codes with k data symbols is obviously intractable in general. For example, in relatively small values of k the probability of a candidate ASSET being "Possibly Balanced" is less than 0.01 in most cases. These probabilities quickly approach 0 as k increases. Second, the effectiveness of the BASSET generation algorithm compared to brute force search is quite apparent from the figures. When the fraction of BASSETs to the total space of ASSETs is effectively 0, as is the case when $k = 16$ and $t = 8$, the GEN-ALL-BASSET algorithm is invaluable.

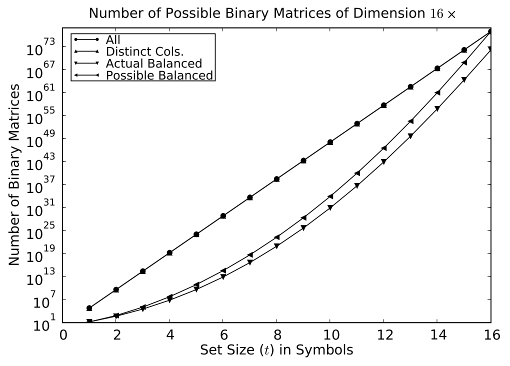
Again, it is assumed that a brute-force algorithm will be primarily used to find BASSETs given a specific code, while the GEN-ALL-BASSET algorithm is apt for building codes around a set of BASSETs. One can use heuristic search to find the appropriate BASSETs in both cases. In the case of the BF-CODE-BASSET algorithm, the total search space for ASSETs of size t is $\binom{n}{t}$. When building a code around BASSETs and using the GEN-ALL-BASSET generation algorithm, the total search space is $\binom{k}{t} \cdot 2^{t^2-t}$. The heuristic search algorithm is entirely dependent on both the underlying system and any



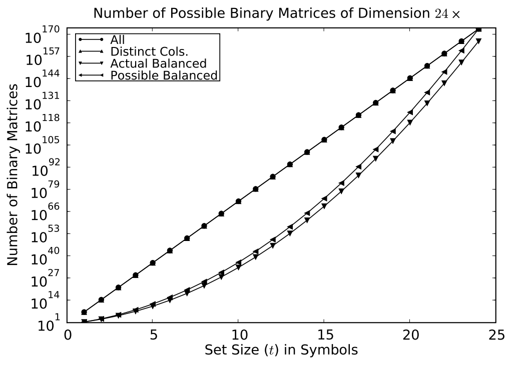
(a)



(b)



(c)



(d)

Figure 8.2: Number of processed binary matrices when searching for BASSETs of size t. An ASSET of size t can be represented by a $k \times t$ matrix, where k is the number of data symbols. Each graph represents the space of codes with 4 (a), 8 (b), 16 (c) and 24 (d) data symbols. The data sets in each graph are: all possible $k \times t$ matrices (All), all possible $k \times t$ matrices with distinct columns (Distinct Cols.), all possible $k \times t$ matrices with $(k - t)$ all-zero rows (Possible Balanced) and all $k \times t$ matrices that correspond to a BASSET.

associated policies.

Our exploration so far has been limited to erasure-coded symbols. Without a concrete system and set of policies dictating how data is read and written, there is no clear way to choose appropriate ASSETs. As an example, if data reads are much more common than writes, keeping most of the parity symbols inactive may be an appropriate policy. While this section treated ASSETs in a general fashion, the remaining sections focus on a specific type of system and policy used to determine the appropriate ASSETs.

8.3 The Single-Data Connected-Parity Policy (SDCP)

The last section covered finding BASSETs among specific erasure codes or over a class of erasure codes for a given number of data symbols or a candidate code. The algorithms used to find BASSETs are general in nature and are system and policy agnostic. In reality, a set of policies is required to choose the appropriate erasure code(s) and ASSETs for a system. Without a policy or set of policies, there is no way to properly choose ASSETs. In this section, we focus on a specific policy for determining ASSETs, called the single-data connected-parity policy (SDCP).

Definition 8.3.1. ***The single-data connected-parity*** *(SDCP) policy represents the minimum number of symbols in a codeword that must be active to service any write request. If the data symbol is the member of a parity equation, the corresponding parity symbol is* ***connected*** *to the data symbol. An ASSET under the SDCP policy will contain an initiating data symbol and its connected parity symbols. ASSETs structured in this way ensure that enough symbols are active to update parity symbols in a timely fashion. Under the SDCP policy, there exist k possible ASSETs for an erasure code. These ASSETs are termed an SDCP-ENSEMBLE.*

Consider an ASSET, $\mathcal{A}$, created from a (k,m)-CODE code, $\mathcal{C}$, using the SDCP policy. Suppose the only available symbols in $\mathcal{C}$ are the symbols in $\mathcal{A}$. Recall, that a symbol can refer to a single bit or an entire device. Here we assume that a symbol will be associated with a large fraction of a device or a whole device. All data writes to $\mathcal{C}$ will be directed at the single available symbol. Since the SDCP policy was used to create $\mathcal{A}$, all dependent parity symbols are also available and all redundant information can be updated without activating any additional symbols.

We call any system that requires dependent parity to be updated without immediately activating or staging additional symbols an *immediate parity update* system. Any ASSET in an immediate parity update system *must* be a superset of an ASSET built using the SDCP policy. Suppose an ASSET, $\mathcal{A}$, is not a superset of an ASSET built using the SDCP policy. This means that $\mathcal{A}$ does not contain all of the connected parity symbols for *any* of the data symbols in $\mathcal{A}$. As a consequence, all data updates will require a symbol activation or the corresponding parity information will have to be staged until the appropriate symbols are active. This conflicts with any immediate parity update system; thus, any ASSET within an immediate parity update system must be a superset of an ASSET created by the SDCP policy.

We focus on storage systems that require immediate parity updates. Based on the aforementioned observations, Property 8.3.2 must hold for all ASSETs.

Property 8.3.2. *The* SDCP *policy represents the* ***minimal active symbol requirement*** *for any immediate parity update system. All* ASSET*s in such a system must be based on the* SDCP *policy. That is, the symbols in an* ASSET *must be a superset of the symbols in an* ASSET *under the* SDCP *policy.*

Property 8.3.2 implies that many of the results that apply to the SDCP policy will apply to immediate parity update systems in general.

Much of the discussion so far has focused on properties of individual ASSETs. In the remainder of this section we analyze properties of ensembles of ASSETs under the SDCP policy. To a system, an ensemble of ASSETs describes how symbols are activated and deactivated as a function of time, workload or some other policy.

We can make general statements about erasure codes when the policy used to determine the ASSETs for a system is fixed. First, we analyze balanced SDCP-ENSEMBLEs. Since the ASSET policy is fixed, creating balanced SDCP-ENSEMBLEs is completely dependent on the structure of the erasure code. Second, we explore the existence of k-balanced SDCP-ENSEMBLEs policy. A k-balanced SDCP-ENSEMBLE contains k ASSETs created under the SDCP policy with a reconstructability of k. In other words every data symbol is always accessible.

We assume that each data symbol contributes to *at least one* parity symbol, otherwise, a data symbol may be unprotected. The m parity elements *cover* all k data elements if each data element contributes to at least one parity equation. The parity symbols of a code are *unique* if no pair of columns in the parity submatrix of the code are equal.

8.3.1 Balanced Available Symbol Set Ensembles

The main result of this section is a bound on the rate of a (k,m)-CODE code that has a balanced SDCP-ENSEMBLE. The only time an (k,m)-MDS code has a balanced SDCP-ENSEMBLE is when $m = k - 1$. When $k \geq 2$, the rate will be at most $\frac{2}{3}$. This same bound, $\frac{k}{k+m} \leq \frac{2}{3}$, is obtained for ($k$,$m$)-NXOR codes.

A specific type of NXOR code, we call a *minimum-cover dual-parity* code (MCDP), will always have a balanced SDCP-ENSEMBLEs. There are two major aspects of a MCDP code: all data symbols must be covered by the parity symbols (minimum-cover) and each parity equation is unique and consists of exactly two data symbols (dual-parity).

In terms of the parity equations, a MCDP code is the most sparse NXOR code that has a balanced SDCP-ENSEMBLE. The sparsity of the code refers to the sparseness of the parity submatrix: the parity submatrix of a MCDP code has the minimum number of non-zero entries for any code that covers all data elements and has a balanced SDCP-ENSEMBLE. Any code that has one or more parity columns with a Hamming weight of 1 will never have a balanced ensemble under the SDCP policy, since such a column represents a mirrored copy of a data symbol. Thus, the columns of the parity submatrix of a code with a balanced SDCP-ENSEMBLE must be unique and have a Hamming weight of at least 2.

Lemma 8.3.3. *Any* NXOR *code that has a parity submatrix with* ***unique*** *columns of Hamming weight* 2 *that* ***cover*** *all of the data symbols (a* MCDP *code) has a balanced* SDCP-ENSEMBLE.

Proof. We prove this by showing that an arbitrary ASSET from a SDCP-ENSEMBLE is balanced.

Consider a (k,m)-NXOR code that has a parity submatrix with unique columns of Hamming weight 2 that cover all of the data symbols. Under the SDCP policy, an

ASSET contains a single data symbol, d_i, and all parity symbols connected to d_i. k ASSETs are created from each of the k data symbols. Let d_i be the data symbol associated with the i-th ASSET, a_i. a_i contains d_i and all parity symbols connected to d_i. The XOR sum of each connected parity column with the column associated with d_i will result in a unique data symbol. That is, if there are m' parity symbols connected to d_i, we can solve for m' different data symbols. We can reconstruct $m' + 1$ data symbols and there are $m' + 1$ symbols in a_i; thus, a_i is balanced. □

We call the SDCP-ENSEMBLE created for a MCDP code a *minimal balanced ensemble*. As we will show, if a balanced SDCP-ENSEMBLE exists for a (k,m)-NXOR code, then a minimal balanced ensemble must also exist for a (k,m)-NXOR code.

A minimal balanced ensemble represents the balanced SDCP-ENSEMBLE built from a (k,m)-NXOR code with a maximally sparse parity submatrix. That is, removing any data symbol from a parity equation will result in a parity equation with one data symbol. This means that *every* (k,m)-NXOR code with a balanced SDCP-ENSEMBLE *must* be created by adding parity dependencies onto a MCDP (k,m)-NXOR code. In other words, every (k,m)-NXOR code with a balanced SDCP-ENSEMBLE is created by flipping 0 entries in the parity submatrix of a MCDP code to 1. This suggests that a balanced SDCP-ENSEMBLE for a (k,m)-NXOR code only exists if a minimal balanced ensemble exists for a (k,m)-NXOR code. Here, we formally prove this statement.

Lemma 8.3.4. *A* ***minimal balanced ensemble*** *under the* SDCP *policy must exist for some (k,m)-*NXOR *code in order for a balanced* SDCP-ENSEMBLE *to exist for any (k,m)-*NXOR *code.*

Proof.

Suppose a balanced ensemble exists for a (k,m)-NXOR code and a minimal balanced ensemble **does not** exist for a (k,m)-NXOR code. If a balanced ensemble exists under the SDCP policy, then each write group must at least be able to to solve for the defining data element and some other data symbol. Additionally, the m parity symbols must cover all k data symbols; thus, a parity submatrix consisting of columns with Hamming weight 2 that covers all of the data symbols can be constructed. This contradicts the claim that a minimal balanced ensemble does not exist. □

The simple structure of a MCDP code allows us to determine the maximal rate of a MCDP code. It follows from Lemma 8.3.4 that this rate is also the maximum rate of any code that has a balanced SDCP-ENSEMBLE.

Theorem 8.3.5. *If $m \leq k \leq 2 \cdot m$, then there exists a balanced* SDCP-ENSEMBLE *for a (k,m)-*NXOR *code. If $k > m \cdot 2$, then a balanced ensemble does not exist under the* SDCP *policy.*

Proof. A MCDP code will have a parity submatrix with exactly $m \cdot 2$ ones that cover all of the data elements. By Lemma 8.3.3, a MCDP code will always have a balanced SDCP ensemble (minimal balanced ensemble). Lemma 8.3.4 states that under the SDCP policy, a minimal balanced ensemble must exist for some (k,m)-NXOR in order for a balanced SDCP-ENSEMBLE to exist for any (k,m)-NXOR. Here, we show that we can always build a MCDP code when $m \leq k \leq 2 \cdot m$ and cannot build a MCDP code when $k > m \cdot 2$.

When $m \leq k \leq 2 \cdot m$, a MCDP code can always be created as follows. The parity submatrix begins as the $m \times m$ identity matrix (I_m) and $(k - m) \times m$ all-zero matrix:

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

If $k = m$, then for each row in the parity submatrix, set the $((i+1) \mod m)$-th entry to 1. If $k > m$, then for $(k - m) \leq i \leq k$, set the $((i + 1) \mod m)$-th entry in row i to 1. This construction will always result in a parity submatrix that covers all data symbols and each column has a Hamming weight of 2: a MCDP code, which has a balanced SDCP ensemble.

If $k > m \cdot 2$, then a MCDP code cannot be created since all k data elements will not be covered and by Lemma 8.3.4 a balanced ensemble cannot exist.

□

Theorem 8.3.5 places a bound on the maximum rate of a (k,m)-NXOR code that has a balanced SDCP-ENSEMBLE. The highest code rate is $\frac{2 \cdot m}{3 \cdot m} = .667$ when $k = 2 \cdot m$. This also matches the bound placed on MDS codes, leading to the following observation.

Observation 8.3.6. *In order to have a balanced ensemble in an immediate parity update system, the rate of the underlying erasure code is bound by* $\frac{k}{k+m} \leq \frac{2}{3}$.

Observation 8.3.6 is quite powerful. Since the ASSETs of a immediate parity update system must be a superset of a SDCP ASSET, the only useful codes—from a power-aware coding perspective—will have a rate of at most $\frac{2}{3}$. Given that the number of parity symbols in an non-mirrored, erasure-coded system tend to never exceed the number of data symbols, the rate of useful codes is bound on both sides by $\frac{1}{2} \leq \frac{k}{k+m} \leq \frac{2}{3}$.

8.3.2 k-Balanced Available Symbol Set Ensembles

From the perspective of a power-managed system, a best case scenario would allow for devices to be deactivated, all parity updates to immediately complete and have all data symbols be available. Such a setup requires an ensemble of k-balanced ASSETs that contain a mixture of data and parity symbols. It turns out that in this setup a MDS code will always have a lower rate than a NXOR code under the SDCP policy. This is not necessarily the case when augmenting the SDCP policy; we show an example of an NXOR that has a comparable rate to a MDS code and has a balanced ensemble.

The only MDS code that has a k-balanced SDCP-ENSEMBLE is a (k,$k-1$)-MDS code. As stated by Property 8.1.4, at least k symbols must be available in order to reconstruct any erased symbol in an MDS code. In an immediate update system, all m parity symbols of an MDS code must always be active. When $m < k - 1$, each ASSET will contain at most $k-1$ symbols, which is insufficient for rebuilding all k data symbols. If $m > k - 1$, each ASSET will contain at least $k + 1$ symbols: too many symbols for a balanced ASSET. It follows that under the SDCP policy, k-BASSETs can only be created when $m = k - 1$.

In order for a (k,m)-NXOR code to have a k-balanced SDCP-ENSEMBLE, $m = k$. If $m < k$, then no k-balanced SDCP-ENSEMBLE exists.

Lemma 8.3.7. *If a (k,m)-*NXOR *code has a k-balanced* SDCP-ENSEMBLE*, then the number of parity symbols must be equal to the number of data symbols.*

Proof. By definition, a BASSET with k symbols can reconstruct k data symbols. Under the SDCP policy, each BASSET must have at least $k-1$ parity symbols. When $m = k-1$, the parity submatrix of the generator matrix is all ones and will never produce a k-balanced SDCP-ENSEMBLE. If $m < k-1$ there is no way to make a k-balanced SDCP ASSET. When $k = m$, $\binom{k}{k-1} = k$ parity columns can be created so that each data symbol is covered by $k-1$ parity symbols. □

Lemma 8.3.7 shows that MDS codes will require less space overhead than NXOR codes when using k-balanced SDCP-ENSEMBLEs. k-balanced ensembles can be created for lower-rate codes by augmenting the SDCP policy. Here we give an example, leaving more general analysis to future work.

Assume a code, C =(4,3)-NXOR and let $s_0, s_1, \ldots, s_6$ be the symbols associated with C. The parity equations of C are:

$$\begin{aligned} s_4 &= s_0 \oplus s_1 \\ s_5 &= s_1 \oplus s_2 \\ s_6 &= s_2 \oplus s_3 \end{aligned}$$

Under the SDCP policy the ASSETs for both codes are balanced, but not k-balanced. The ASSETs under the SDCP policy for C are:

$$\{s_0, s_4\}$$
$$\{s_1, s_4, s_5\}$$
$$\{s_2, s_5, s_6\}$$
$$\{s_3, s_6\}$$

The algorithm described in Section 8.2.2.1 can be used to find k-BASSETs such that the BASSETs are subsets of the SDCP ASSETs. The following ASSET ensemble for C is k-balanced:

$$\{s_0, s_3, s_4, s_6\}$$
$$\{s_1, s_3, s_4, s_5\}$$
$$\{s_0, s_2, s_5, s_6\}$$
$$\{s_3, s_4, s_5, s_6\}$$

This code has the same rate as an MDS code having a k-balanced SDCP-ENSEMBLE. There is an obvious disadvantage to this approach. The MDS code with the same parameters will provide higher fault tolerance: (4,3)-NXOR is 1 symbol fault tolerant, while (4,3)-MDS is 3 symbol fault tolerant. On the other hand, the use of NXOR codes allows a system designer to effectively cycle the devices. For example, the 3 parity symbols of a (4,3)-MDS code must always be active; this is not necessarily the case for NXOR codes. The use of NXOR codes provides more freedom in how ASSETs are created.

8.4 Architecture of a Power-Aware Storage System

Here we present the architecture of a power-aware storage system and relevant terms specific to this architecture. This architecture is quite similar to Pergamum [58] and serves as a straw-man system for presenting the power-aware coding techniques.

At a high-level, all writes into the system are served in a log-structured manner. In log-structured file system terminology, the current segment of the log corresponds to one or more devices. This means that only the devices associated with the current segment need to be active to service the write workload. This is an immediate parity update system; thus, parity updates will never be staged and later moved to their respective persistent stores. As a result, the parity devices associated with the current segment must also be active to ensure all parity information is updated as soon as possible.

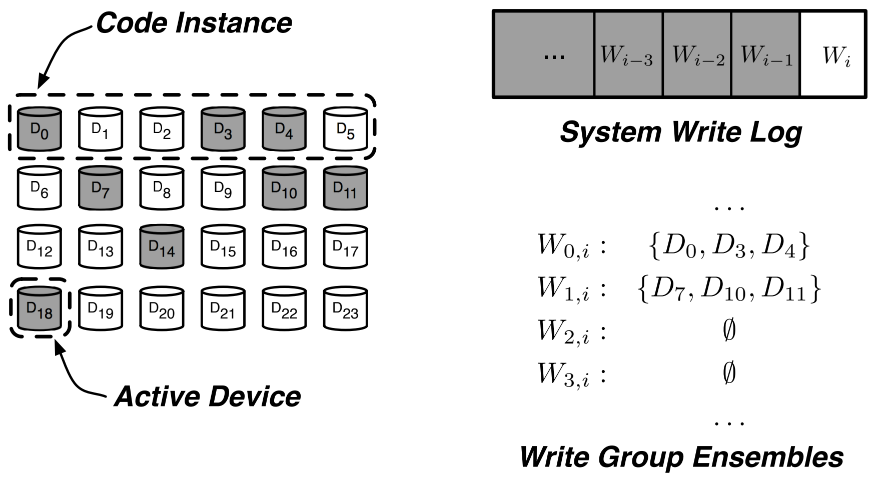

Code Instance
D_0 D_1 D_2 D_3 D_4 D_5
D_6 D_7 D_8 D_9 D_{10} D_{11}
D_{12} D_{13} D_{14} D_{15} D_{16} D_{17}
D_{18} D_{19} D_{20} D_{21} D_{22} D_{23}
Active Device
... W_{i-3} W_{i-2} W_{i-1} W_i
System Write Log
...
$W_{0,i}$: $\{D_0, D_3, D_4\}$
$W_{1,i}$: $\{D_7, D_{10}, D_{11}\}$
$W_{2,i}$: $\emptyset$
$W_{3,i}$: $\emptyset$
...
Write Group Ensembles

Figure 8.3: High-level view of a power-managed system, where storage devices are deactivated to save power. The devices are partitioned into code instances. Each code instance will have a write group ensemble, which determines how devices are activated to serve an incoming write workload. All writes into the system are done so in a log-structured manner. The current segment of the log is associated with a write group from each code instance. A write group can be the empty set, which means that none of the devices in the code instance are used to serve the current write workload.

Figure 8.3 shows a system that contains a number of storage devices that may be deactivated to save power. Each device in the system is partitioned into equal-sized *disklets*. A disklet can be anything from a single sector up to an entire device. The system pictured in Figure 8.3 has whole-device-sized disklets. The disklets in the system are further partitioned into *code instances*. A code instance is a one-to-one mapping of erasure code symbols to disklets.

A *write group* is a set of devices in a code instance that are active to service a write workload. A *write group ensemble* is the set of all write groups for a code instance. Individual write groups and the resulting write group ensemble depends on factors such as expected workload, reliability and power budget. A write group is distinct from an ASSET in that it refers to devices, not symbols.

The write group ensemble will be generated based on an ASSET ensemble. The current write group for a code instance is determined by an ASSET. The devices associated with an ASSET will be the devices in a write group. Here we assume a one-to-one mapping of devices to code symbols, thus the terms ASSET and write group are equivalent.

The current segment of the log will be written to devices from one or more write groups. In turn, the current segment may correspond to devices in different code instances.

The system shown in Figure 8.3 partitions the devices into four, disjoint 6-disk code instances:

$$\{D_0, D_1, D_2, D_3, D_4, D_5\}$$
$$\{D_6, D_7, D_8, D_9, D_{10}, D_{11}\}$$
$$\{D_{12}, D_{13}, D_{14}, D_{15}, D_{16}, D_{17}\}$$
$$\{D_{18}, D_{19}, D_{20}, D_{21}, D_{22}, D_{23}\}.$$

Suppose the current segment in the log corresponds to data disk D_0 from the first code instance and data disk D_7 from the second code instance. Furthermore, assume that devices D_3 and D_4 contain the parity symbols that are connected to data disk D_0 and devices D_{10} and D_{11} contain the parity symbols that are connected to data disk D_7. Then, the current segment will be written across two write groups: $\{D_0, D_4, D_7\}$

and $\{D_3, D_{10}, D_{11}\}$.

Of the 24 devices shown in Figure 8.3, 8 devices are active. The active devices are shaded in gray, while the inactive devices are white. Six of the active devices—$D_0, D_3, D_4, D_7, D_{10}, D_{11}$—are called *deterministic activations* because they represent the write groups associated with the current segment. The remaining active devices—D_{14} and D_{18}— are called *transient activations.* A transiently active device is a device that is activated to service a read request and is typically deactivated after some threshold.

Three conditions are necessary for an erasure-coded system to be power-aware. First, a read policy is necessary to dictate if data is accessed directly off a disk or reconstructed using redundant information. Second, the system must have policies that service writes in a way that minimizes operational power consumption while maintaining a sufficient level of reliability. Finally, when disk activation is necessary to service a read request, a policy is needed to determine how to efficiently schedule disk activations.

8.4.1 Servicing Reads

Read requests are satisfied by either accessing an active disk or using the erasure code to reconstruct the appropriate content from the active disks. If the information provided on the active disks is insufficient for serving the request, other disks must be activated. This is a *transient disk activation.* A transient activation may be used to directly service the request or as part of data reconstruction if the request involves multiple disks. Since a transient activation involves a disk that is not a member of an active write group, it will be deactivated after some fixed period and will not service writes. In addition to reads, transient activations may also be used to perform background operations such as disk scrubbing [56].

Choosing to perform reconstruction, transient activation, or a combination of the two depends on the situation, environment and workload. There may be cases where a transient activation may be more power efficient than reconstructing the data from active disks. The system should optimize for each read request based on the state of the system and number of disks involved.

Most systems handle read requests to inactive disks using the *naive strategy*, which simply activates the disks involved in the request. Multiple transient activations can have a dramatic effect on system power consumption and reliability. In addition, recent analysis shows that the system reliability will decrease if disks are power cycled too often [56]. In order to minimize power consumption and maintain a reasonable level of reliability, the system should minimize the number of transient disk activations.

8.4.1.1 Power-Aware Read Algorithm

At a high level, the power-aware read algorithm treats symbols on inactive devices as erased and relies on matrix methods (elementary matrix operations) to determine if partial or whole-stripe reconstruction is possible using disks that are already active [24]. If reconstruction of any erased data is possible, the matrix transformations result in appropriate recovery equations. Instead of marking a disk as failed (or erased), we mark all symbols on inactive devices *tentatively lost*. A tentatively lost symbol is made available through activation. When a read request involves data that is tentatively lost, we try to reconstruct the symbols in a way that minimizes the number of disk activations.

Our read algorithm relies on a function that determines if lost data is recoverable, and if so, the equations needed to reconstruct. The recovery equations for tentatively lost data are computed using the underlying generator matrix, G. A matrix, G', is constructed by zeroing out the columns in G that correspond to the tentatively lost symbols. In order to determine the recovery equations we must find a pseudo-inverse, R (as defined by Hafner *et al.* [24]), of G'. Suppose the vector c' is the vector $c = d \cdot G$ with zeroes in the positions corresponding to tentatively lost symbols. Then $c' \cdot R = d'$, where the non-zero elements of d' are the corresponding recoverable symbols of d and the zero elements are unrecoverable. In this case, R contains the recovery equations and c' contains the available data and parity symbols.

The power-aware read algorithm is shown in Algorithm 6. The algorithm takes the inactive symbols involved in the read request (I), the generator matrix for the underlying code (G) and the set of currently inactive symbols (L) as input. The function `recoverable` uses the aforementioned matrix methods to determine the symbols that are recoverable based on the underlying code and the set of currently available (or recoverable) symbols. The `recoverable` function returns a list of recoverable symbols, L', and the corresponding recovery equations. The `activate_disk` function, which is explained in Section 8.4.1.2, determines the disk (or disks) to activate given the read request and state of the system.

As an example, assume a one-to-one mapping of code symbols to devices. Suppose $I = \{s_2, s_4\}$ and $L = \{s_1, s_2, s_3, s_4\}$ in the code instance shown in Figure 8.1. In the first iteration, `recoverable` returns $(\{s_4 = s_0 \oplus s_5 \oplus s_6 \oplus s_7\}, \{s_4\})$. The second iteration begins with $I = \{s_2\}$ and $L = \{s_1, s_2, s_3\}$ and `recoverable` returns $(\emptyset, \emptyset)$, therefore, a disk must be activated. Since s_2 is the only disk left in I, `activate_disk` returns $(\{s_2 = s_2\}, \{s_2\})$; disk D_2 must be activated. The loop invariant evaluates to `false` at the beginning of the third iteration and the algorithm returns the corresponding recovery equations.

Algorithm 6 Recover I using G and L

```
1: while $I \neq \emptyset$ do
2:     (eqns, $L'$) ← recoverable($G$, $L$)
3:     if $L' = \emptyset$ then
4:         (eqns, $L'$) ← activate_disk($L, G, I$)
5:     else
6:         $I \leftarrow I - L'$
7:         $L \leftarrow L - L'$
8:     end if
9:     all_eqns.append(eqns)
10:    return all_eqns
11: end while
```

8.4.1.2 Disk Activation Algorithm

Since our approach takes advantage of the underlying erasure code, there exist many cases where the naive activations can be avoided. If the read request contains a single inactive symbol that cannot be reconstructed, then we simply activate the corresponding disk. If more than one inactive symbol is in a read request, we must determine the minimum number of activations required to service the request.

Algorithm 7 performs a brute force search of potential disks to activate by generating all possible combinations of disk activations (i.e. powerset of inactive disks). All disks involved in the read request are in the set I, while all inactive disks are in the set L. The powerset function, $\mathcal{P}$, orders the combinations in ascending order by size. For each combination, s, the algorithm determines if the request can be satisfied when the disks listed in s are activated (via `is_fully_recoverable`). It is assumed that the `is_fully_recoverable` function has access to the symbol-to-disk mapping. Once a satisfactory combination is chosen, the algorithm returns the disks to activate and an updated list of inactive devices. Since the combinations are ordered, this algorithm will return the minimum number of activations needed to service the request.

Algorithm 7 Determine the minimum number of disk activations required to service request I when disks in L are inactive.

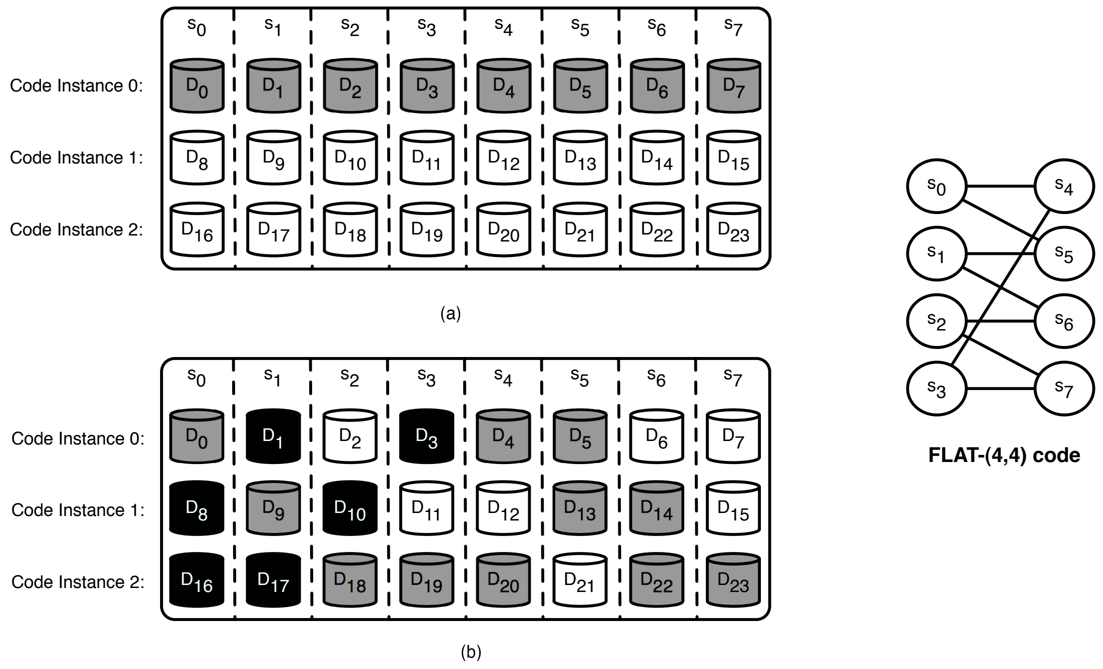
```
1: for $s \in \mathcal{P}(L) - \emptyset$ do
2:     try ← L − $s$
3:     if is_fully_recoverable(try,$I$,$G$) then
4:         $L \leftarrow L - s$
5:         return $(s, L)$
6:     end if
7: end for
```

8.4.2 Servicing Writes

The total power consumed by the storage system is heavily dependent on the write group ensemble. A write group will likely be active for a number of hours and keep the number of active disks to a minimum. In addition, the number of active disks determine which data can be reconstructed; thus, a proper balance is required to service both writes and reads into the system.

Given the current assumptions, two types of system designs stand out for a power managed system. Both of these designs are shown in Figure 8.4. Assume that the incoming write workload can be handled by 4 data devices, where each block of user data corresponds to 1 data block and 2 parity blocks.

Figure 8.4-(a) shows a design where whole code instances are kept active to



Code Instance 0:
Code Instance 1:
Code Instance 2:
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7
D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
D_8 D_9 D_{10} D_{11} D_{12} D_{13} D_{14} D_{15}
D_{16} D_{17} D_{18} D_{19} D_{20} D_{21} D_{22} D_{23}
(a)
(b)
FLAT-(4,4) code

Figure 8.4: Two possible setups for a power-managed system that satisfies our three assumptions. The first setup (a), sends all write requests to a single code instance and services read requests by spinning up inactive devices. The second setup (b), activates subsets of each code instance to service the write workload and handles read requests via partial reconstruction or by spinning up inactive devices. Each code instance is defined by the (4,4)-FLAT code shown on the right.

service a write workload. In this case, a single code instance is active and the remaining code instances are inactive. The active disks are shaded. Any read request to the disks in code instance 0 can be serviced without activating any additional devices; read requests to all other disks will require activations.

Another strategy is to activate data devices across multiple code instances in a way that can handle the incoming write workload. In this case, a subset of devices in each code instance are activated to handle the write workload. Using the (4,4)-FLAT code shown in Figure 8.4, each write to a data device requires two updates to parity devices; thus, activating any 4 data devices across the code instances will be able to handle the workload. Again, the active devices are shaded. A single data device and its dependent parity devices are activated in the first two code instances. Two data devices and their dependent parity are activated in the last code instance. Since we have spread the write workload across multiple code instances, most read requests need not be serviced by activating a disk. The disks corresponding to solvable data are shaded in black.

Code instances 0 and 1 correspond to 3-BASSETs under the SDCP policy. Three devices are active in each code instance to serve an incoming write workload. Code instance 0 has devices D_0, D_4 and D_5 active and can reconstruct the contents of data devices D_0, D_1 and D_3. Code instance 1 has devices D_9, D_{13} and D_{14} active and can reconstruct the contents of data devices D_8, D_9 and D_{10}. Code instance 2 corresponds to an ASSET that is not balanced: 5 devices are active, but all 4 devices worth of data can be read without activating an additional device. There is no way to serve the incoming workload under this configuration and have all code instances correspond to balanced ASSETs, so this configuration maximizes the number of BASSETs with two.

The main disadvantage to activating devices as shown in Figure 8.4-(b), is the power consumed when there is a very light read workload. Any savings via partial reconstruction are overshadowed by the disks active to service the write workload. This motivates a direct tradeoff. There are 8 active devices in Figure 8.4-(a), which is capable of handling the write workload, but will require disk activations for read requests to 8 of the 12 data disks. The setup in Figure 8.4-(b) requires 11 active devices, but only requires a disk activation for read requests to 2 of the 12 data disks.

8.5 Evaluation

In this section we perform an elementary tradeoff analysis. Using the HFR Simulator, we compare the estimated reliability of a variety of codes to their relative power savings. We rely on three metrics for estimating the relative power savings: aggregate write group size, maximum reconstructability and relative gain. The aggregate reconstructability of a code describes the average number of data symbols that can be recovered when there are no transient activations. The relative gain of a code estimates the effect of performing partial reconstruction over simply activating the target disks.

We assume a system with an architecture similar to the system described in Section 8.4. Here we focus on the performance of a single code instance, where the mapping of code symbols to devices is one-to-one; thus, the number of devices in a code instance is equal to the number of code symbols. While the metrics are derived in terms of write groups, they can be easily generalized to ASSETs. Finally, we assume the system uses the SDCP policy for creating write groups (or ASSETs).

8.5.1 Metrics

We have developed three metrics to measure the aggregate write group size based on a write group policy and potential disk activation avoidance by reconstructing data on inactive devices for a single code instance. The *aggregate write group size* metric is rather straightforward: compute the average number of disks active over a write group ensemble for a code instance. We provide two metrics for measuring potential disk activation avoidance given a code instance and write group ensemble. First, *aggregate reconstructability* measures the average number of reconstructable data symbols over all write groups within a write group ensemble. In other words, aggregate reconstructability captures the average number of data symbols that can be accessed without transient activations. Second, *aggregate relative gain* measures the impact of using data reconstruction to access data on inactive devices. Device activations are counted over every possible read request combination to a code instance. The relative gain metric is based on the ratio of device activations for data reconstruction and the naive read strategy.

Each metric is normalized to facilitate comparison between different codes and code instance configurations.

8.5.1.1 Aggregate Write Group Size

Given the write group ensemble $W = \{w_0, w_1, \ldots, w_{k-1}\}$, the average number of active devices used to service write requests—assuming the duration of each write group is equal—is

$$\frac{1}{k}\sum_{i=0}^{k-1}|w_i|$$

This value can be normalized with respect to the total number of symbols, n. The normalized version is calculated as

$$1 - \frac{\frac{1}{k}\sum_{i=0}^{k-1}|w_i|}{n}.$$

The normalization results in the following scale: 0 means all of the devices within a code instance are active over the write group ensemble and a value of 1 corresponds to $W = \emptyset$. From a purely power consumption standpoint, a value of 1 is the best and a value of 0 is the worst.

8.5.1.2 Aggregate Reconstructability

By Definition 8.1.3, the reconstructability of a code C, $R(C, s)$, is defined as the number of readable data symbols when symbols in s are available. The average reconstructability for a code C with write groups in W is calculated as

$$R_{\text{avg}}(C, W) = \frac{1}{k}\sum_{i=0}^{k-1} R(C, w_i).$$

Suppose there are two codes C and C' that have k and k' data symbols, respectively. When $k \neq k'$, $R_{\text{avg}}(C, W)$ and $R_{\text{avg}}(C', W')$ cannot be compared in an apples-to-apples fashion. In order to facilitate a fair comparison between codes, average reconstructability is normalized to give us the aggregate reconstructability

$$R_{\text{agg}}(C, W) = \frac{R_{\text{avg}}(C, W)}{k}.$$

As with aggregate write group size, aggregate reconstructability normalizes to a scale between 0 and 1. A value of 0 corresponds to a code and write group ensemble pair that cannot reconstruct any data symbols. A value of 1 corresponds to a (C, W) pair where all k data symbols are reconstructable for every write group in W.

8.5.1.3 Aggregate Relative Gain

The read strategy in most power-managed systems will always energize inactive disks that receive read requests; this is the naive strategy. Avoiding spin-up by recovering contents of inactive disks is called the recovery strategy. Relative gain is used to compare any two arbitrary read strategies. Here, we compare the naive and recovery strategies.

Consider a code instance containing n disks, labeled $0, 1, ..., n-1$, with k data disks and $n-k$ parity disks. Let $D = \{0, 1, ..., k-1\}$ be the set of data disks, $P = \{k, k+1, \ldots, n-1\}$ be the set of parity disks and $A \subseteq (D \bigcup P)$ be the set of active disks. Every possible read request combination with respect to the devices can be described by the powerset $(\mathcal{P})$ of the k data disks. Every possible read request to the code instance is contained in the set $C = \mathcal{P}(D)$. For some $c \in C$, $c' = c \setminus A$ represents the inactive devices involved in a read request. In the naive read strategy, $|c'|$ disk activations are required to service the read request.

Define $\beta(c, A)$ as the function described by Algorithm 7 (cf. Section 8.4.1.2). $\beta(c, A)$ returns the minimal number of disk activations required to service the read request, c, when A devices are available. The number of such activations is given by $|\beta(c, A)|$.

The write group ensemble completely describes the devices that will be active at some point in time to service write requests. Here we ignore any time component and only consider the number of write groups and each groups associated devices. Here, we consider the SDCP policy, therefore, there are k total write groups: $W = \{w_0, w_1, \ldots, w_{k-1}\}$. Each w_i is a subset of $(D \bigcup P)$ and describes the active devices for the write group's scheduled slot.

The relative gain metric computes the ratio of device activations when performing partial reconstruction to simply activating devices for a single write group

$$G_i(C) = \sum_{c \in C} \left(1 - \frac{|\beta(c, w_i)|}{|c \setminus w_i|} \right)$$

The relative gain across all write groups in an ensemble is simply computed as the average across all write groups

$$G(C) = \frac{1}{k} \sum_{i=0}^{k-1} G_i(C).$$

If the naive disk activation policy perfectly matches partial reconstruction, then the ratio in each $G_i(C)$ will be 1 and the relative gain will be 0. Similarly, when partial reconstruction requires no activations over all possible requests, then the relative gain will be 1.

8.5.2 Setup

The codes used to analyze the power-space-reliability tradeoff are shown in Table 8.1. Given the exploratory nature of this chapter, these results serve more as a catalyst for more discussion and work in power-aware coding than a means of generating concrete recommendations to those interested in using the power-aware coding techniques. In short, defining the metrics for such a tradeoff and showing that a tradeoff does, in fact, exist is a substantial contribution; this rudimentary analysis uses the proposed metrics to provide a way to think about the impact of power-aware coding in an erasure-coded system.

For simplicity, the write group ensembles for each code are created using the SDCP policy: each write group corresponds to a single data symbol and its connected parity elements.

Table 8.2 contains the structure and fault tolerance information for the irregular codes used in the tradeoff analysis. Each (k,m)-MDS code is a flat MDS code that can tolerate up to m symbol failures. In this analysis, the codes were chosen based on the total number of symbols, fault tolerance, rate and potential power savings. In addition, we assume that a single symbol is mapped to one and only one device and the number of symbols is equal to the number of devices.

code	Hamming distance	rate	number of disks
(5,5)-BALANCED	3	0.50	10
(5,5)-RAID 10	2	0.50	10
(5,5,2)-WEAVER	3	0.50	10
(7,3)-MDS	4	0.70	10
(8,2)-MDS	3	0.80	10
(9,1)-MDS	2	0.90	10
(8,4)-BALANCED	2	0.67	12
(8,4)-MAX	2	0.67	12
(8,4)-MDS	5	0.67	12
(10,2)-MDS	3	0.83	12
(11,1)-MDS	2	0.92	12
(5,3)-FLAT	2	0.63	8
(4,4)-FLAT	4	0.50	8
(6,2)-MAX	2	0.75	8
(5,3)-MDS	4	0.63	8
(6,2)-MDS	3	0.75	8
(7,1)-MDS	2	0.88	8

Table 8.1: List of erasure codes used in our tradeoff evaluation.

code	Fault Tolerance Vector	parity bitmap
(5,5)-BALANCED	$(0.00, 0.00, 0.00, 0.00, 0.05, 0.30, 1.00)$	$(15, 29, 27, 23, 30)$
(5,5)-RAID 10	$(0.00, 0.11, 0.33, 0.62, 0.87, 1.00)$	$(1, 2, 4, 8, 16)$
(5,5,2)-WEAVER	$(0.00, 0.00, 0.04, 0.19, 0.52, 1.00)$	$(17, 3, 6, 12, 24)$
(8,4)-BALANCED	$(0.00, 0.18, 0.51, 0.84, 1.00)$	$(3, 12, 48, 192)$
(8,4)-MAX	$(0.00, 0.15, 0.43, 0.74, 1.00)$	$(255, 253, 251, 247)$
(5,3)-FLAT	$(0.00, 0.04, 0.29, 1.00)$	$(7, 11, 29)$
(4,4)-FLAT	$(0.00, 0.00, 0.00, 0.20, 1.00)$	$(7, 11, 13, 14)$
(6,2)-MAX	$(0.00, 0.39, 1.00)$	$(63, 62)$

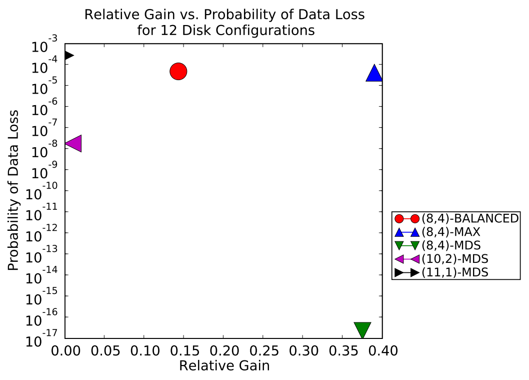
Table 8.2: Fault tolerance and structure of erasure codes used in the tradeoff evaluation.

All of the listed MDS codes obviously provide optimal fault tolerance for the total number of symbols and rate. In the case of MDS codes, in order to reconstruct the data from any inactive device (or symbol), at least k devices must be active. The XOR-based codes, while providing suboptimal fault tolerance, may have the ability to reconstruct data from an inactive device while having less than k active devices. Here, three types of XOR-based codes are considered. Three of the codes exhibit balanced write group ensembles (equivalently BASSETs) under the SDCP policy: (5,5)-BALANCED (5-balanced ensemble), (5,5,2)-WEAVER (2-balanced ensemble) and (8,4)-BALANCED (2-balanced ensemble). Two of the codes provide maximum reconstructability under the SDCP policy: (8,4)-MAX (0.4531 normalized reconstructability) and (6,2)-MAX (0.31 normalized reconstructability). The remaining XOR-based codes were chosen as the most fault-tolerant for a (k,m)-FLAT code: (5,3)-FLAT (1 symbol fault tolerant) and (4,4)-FLAT (3 symbol fault tolerant).

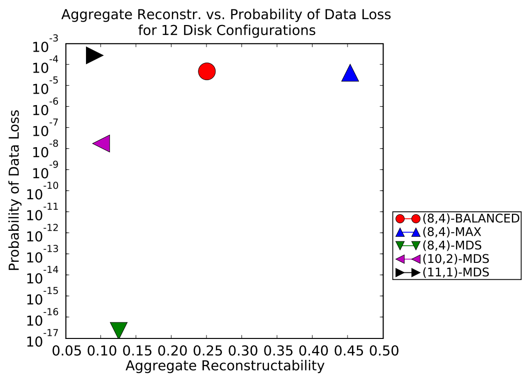
8.5.3 Power-Space-Reliability Tradeoff

Figures 8.5 , 8.6 and 8.7 each contain three graphs and contain comparisons for the 12-disk codes, 10-disk codes and 8-disk codes, respectively. The x-axis of each graph corresponds to the probability of data loss given a 10 year mission time using the HFRS v.2 simulator and the parameters used in Chapter 4. Each graph labeled with an (a) compares relative gain, which corresponds to the number of avoided disk activations under the SDCP policy, with reliability. Each graph labeled with an (b) compares aggregate reconstructability, which corresponds to the number of reconstructable inactive disk reads per write group under the SDCP policy, with reliability. Finally, each graph labeled with an (c) compares the normalized write group size, which corresponds to the average number of active disks used to service writes under the SDCP policy, with reliability.

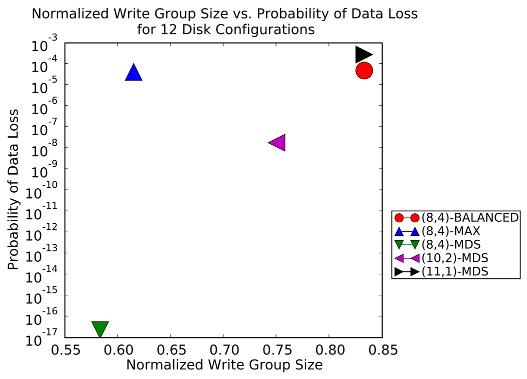
Figure 8.5 contains the 12-disk configurations. As expected the (8,4)-MDS code provides the highest level of reliability across all codes. The (8,4)-MAX provides the highest expected reconstructability and relative gain across all codes. Again, (8,4)-MAX has the maximum reconstructability for any XOR-based code with 8 data and 4 parity, so this is expected. Unfortunately, the (8,4)-MAX code requires too many



(a)

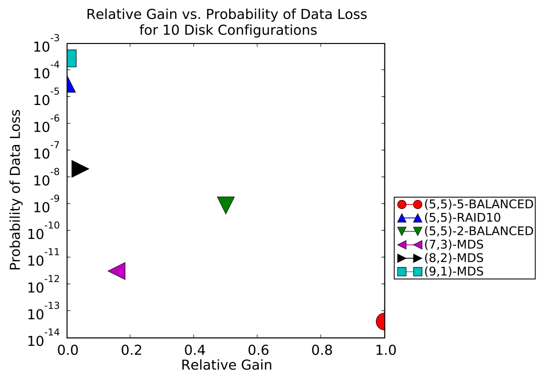


(b)

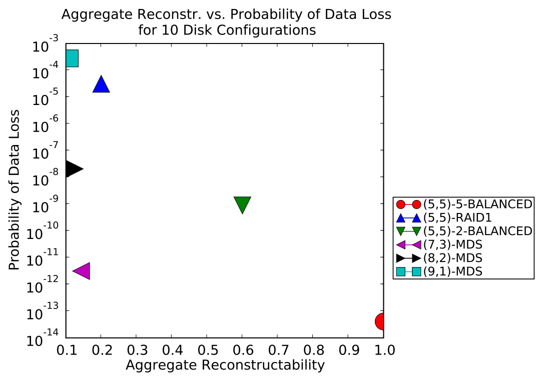


(c)

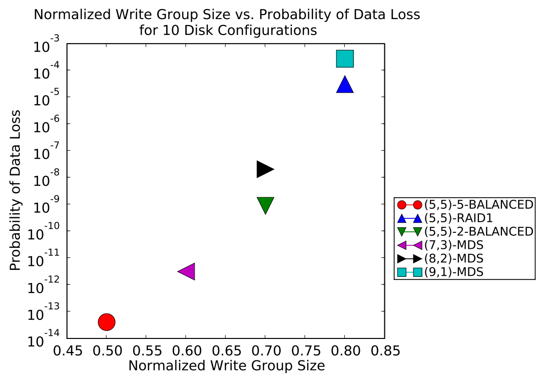
Figure 8.5: Power-reliability tradeoff for 12-disk configurations.



(a)



(b)



(c)

Figure 8.6: Power-reliability tradeoff for 10-disk configurations.

devices for serving writes. We find that higher rate and/or lower density codes typically require fewer additional active devices to service writes, but provide low reliability. When comparing across all codes, no code provides the best space efficiency, reliability and expected reconstructability. It appears that the code with a 2-balanced ensemble provides a decent middle ground between codes. The (8,4)-BALANCED code provides roughly the same reliability as (8,4)-MAX, requires very few devices (2 on average under the SDCP policy) to service writes and provides an expected disk activation savings between the worst and best across the compared codes.

Figure 8.6 shows the tradeoff between 10-disk configurations. Again, no code appears to provide best space efficiency, reliability and expected reconstructability. In fact, for some codes, there appears to be an inverse relationship between expected power due to writes and the reconstructability metrics. The (5,5)-BALANCED code provides the best reliability and reconstructability for all codes. The same code requires the most active devices to service writes, which is most likely due to the density of the code (each parity equation is computed from 4 data symbols). The (9,1)-MDS and (5,5)-RAID 10 codes have the worst reliability and provide the least utility in terms of reconstructability, but require less active devices to service writes under the SDCP policy.

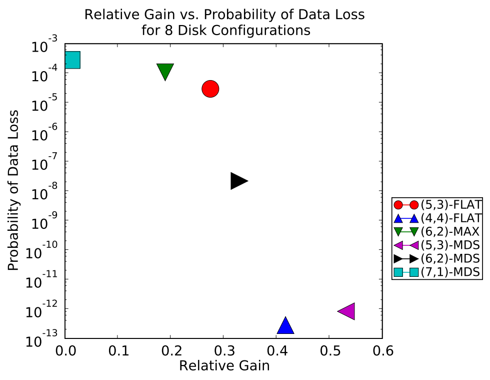
As shown in Figures 8.6 (a) and (b), code rate has an effect on reconstructability and relative gain. Two of the three rate 0.50 codes perform better than all MDS codes. A second order effect is apparent in the structure of each code of rate 0.50. The least sense code ((5,5)-RAID 10) has reconstructability comparable to the higher rate MDS codes. As more dependencies are added to create (5,5)-BALANCED and (5,5,2)-FLAT the reconstructability improves. (5,5)-BALANCED is the most dense and has the best performance in terms of reconstructability.

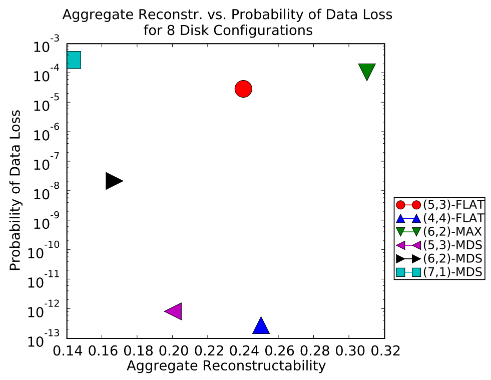
Finally, as seen across all graphs in Figure 8.6, the (5,5,2)-FLAT appears to provide a middle ground between all codes. Under the SDCP policy, this code has a 2-balanced ensemble, its regular structure results in a decent relative gain and on average only 3 devices are active to service writes.

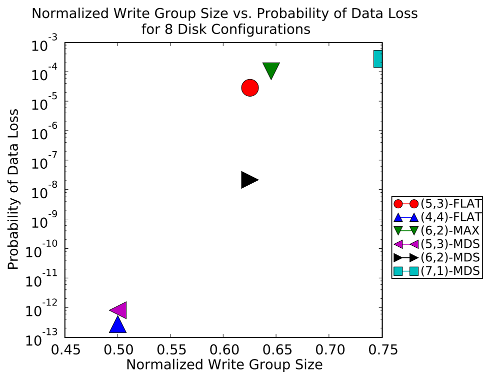
Finally, Figure 8.7 contains the tradeoffs between 8-disk configurations. As with the other code configurations, no code appears to provide best space efficiency, reliability and expected reconstructability. In terms of high reliability and space efficiency, the (4,4)-FLAT code almost rivals the (5,3)-MDS. Both require half of the symbols to be active to service writes under the SDCP policy. The irregularity of (4,4)-FLAT results in higher reliability than the (5,3)-MDS code. While the relative gain of the (5,3)-MDS code is slightly better than the (4,4)-FLAT code, we see comparable reconstructable reads out of the (4,4)-FLAT code.

8.5.4 Discussion

While it is difficult to make any sweeping conclusions based on this rudimentary analysis, a few general statements can be made. Obviously, code rate has a dramatic effect on the reconstructability of a code under power-aware coding. Given two properly constructed codes, the code with a lower rate is expected to provide higher







(c)

(a) (b)

Figure 8.7: Power-reliability tradeoff for 8-disk configurations.

reconstructability. This comes at a cost. The lower rate and high reliability (i.e. higher density) codes tend to require more active devices to service requests under the SDCP policy. This is expected to hold as policies derived from the SDCP are created.

While not true in general, the aggregate reconstructability appears to be correlated with relative gain; thus, a code with high aggregate reconstructability is expected to also have a high relative gain. In the end, the trick is to find a code that has good enough reliability and provides good reconstructability, while not requiring too many additional active symbols to service writes. This implies that NXOR, codes may be well-suited for power-aware coding.

8.6 Conclusion

This chapter has introduced a novel way to save power in erasure-coded systems, called power-aware coding. At a high level, power-aware coding is based on a generalization of traditional fault-tolerance, called reconstructability. Reconstructability uses information on active devices to reconstruct user data on inactive devices, allowing data availability in the face of inactive devices.

The reconstructability metric is used to find sets of active symbols (or devices) that equalize the number of available user data symbols, called balanced. Algorithms were developed to find balanced sets given a candidate erasure code and to build erasure codes around balanced sets.

We have defined a system property called immediate parity update, which enforces timely parity updates and a policy that ensures enough code symbols are active to satisfy the property. We have placed a bound on the rate of any code that can achieve the balanced property in an immediate parity update system.

A high-level system architecture of an immediate parity update system is proposed and subjected to a rudimentary space-power-reliability tradeoff analysis. We have developed three metrics, which are used to quickly estimate the expected power savings of an erasure code and set of activation and access policies. While the metrics do not provide absolute results on the power savings of any particular system, they give a system designer a way to compare a set of candidate system configurations.

Chapter 9

Future Work

9.1 High-Fidelity Reliability (HFR) Simulator

While the techniques presented in Chapter 4 are useful to a researcher, they provide little utility to the practitioner. Currently, both HFRS v.1 and HFRS v.2 are not in a releasable form. In the future, both versions will be merged into a single version, cleaned up and released as an open source Python package. The `markov_model` package and specialized modeling language will also be released along with the updated HFR Simulator. This package will give storage researchers and practitioners a single module that can be used to perform trending analysis and comparison between system instances.

In addition to merging the current versions of the HFR Simulator, there exist two possible major improvements over what exists. First, we found that around and beyond 4-disk fault tolerance, the use of balanced failure biasing alone for non-Markovian systems may not be sufficient. One possible improvement is to augment balanced failure biasing with multi-level splitting techniques [17]. Second, while both versions of the HFR Simulator effectively map one or more code symbols to each device, the current implementation is not efficient for simulating distributed storage systems. Our hope is to accommodate simulations involving hundreds or thousands of devices, with multiple code instances across the devices.

Another important research question is the expected impact of a data loss event. In the future, we plan to flesh out this metric, which can be used in conjunction with the results of Chapter 7 to estimate the impact of a data loss event on different classes of data (i.e. highly sensitive data, re-computable data, etc.).

9.2 Fragment Placement

The redundancy placement problem and associated metrics have applications outside of exploring the effect of fragment placement on reliability. There are two applications that can make use of the results presented in Chapter 7. First, there has been a great deal of buzz surrounding flash+disk hybrid systems. Since flash and disk have different reliability characteristics, the placement of erasure-coded fragments will affect system reliability. Second, certain organizations may store various classes of data. For simplicity, assume that there are two classes of data: sensitive and insensitive. An example of sensitive data may be customer transactions or plans for a space shuttle; in other words, heads will roll if data is lost. Insensitive data could be email spam or a temporary directories. The lost of insensitive data should not affect daily business operations. The placement of such data onto heterogeneous devices presents an interesting research problem, especially when there are many classes of data.

One possible improvement to the techniques presented in Chapter 7 is to exploit the structure of the MEL to make placement decisions. That is, instead of using an aggregate availability measure (RME) for each placement in stochastic optimization, we would like to find ways to directly use the MEL in placement decisions.

9.3 Reconstructability

The idea of reconstructability is quite similar to the reconstruction of lost data presented by Hafner [24]. In short, reconstructability measures the fraction of data symbols that can be recovered given a subset of the code symbols are available. In many cases, all of the data symbols can be recovered. What happens if only a subset can be recovered? What are the consequences/impact of such a scenario? These questions unify some of the ideas presented in Chapters 4 and 7. An interesting research topic would be to unify the expected impact of data loss (cf. Chapter 4), placement of multiple classes of data (implied from Chapter 7) and reconstructability (cf. Chapter 8).

9.4 Power-Aware Coding

Chapter 8 defined the terminology for power-aware coding, presented theoretical results, motivated continued work through a basic reliability-power-space tradeoff analysis and suggested possible architectures where power-aware coding may prove fruitful. Without a workload-based analysis, there is no way to determine how well power-aware coding will fare in the wild. In addition, a proper system-level analysis will be required. One of the looming questions is if power-aware coding is useful enough to incorporate into systems that already rely on architectural techniques such as caching, the use of flash, or the use of other MAID-like techniques.

While the theoretical results were generalized to code symbols and are not necessarily tied to a specific layout of code fragments to devices, much of the analysis assumed that a single code symbol is mapped to a single device, and vice-versa. This policy was used to simplify the analysis. Based on our theoretical results, it is possible that a system can benefit from power-aware coding when mapping a single code symbol to a single device. After performing the simple analysis, we found that the techniques may be more effective in a more general setting that maps multiple symbols to a single device. Such architectures are considered in this section. Additionally, we found a specific code construction that leads to balanced ensembles. As the number of code symbols approaches infinity, the code rate approaches the bound implied by Theorem 8.3.5.

The main purpose of this section is to encourage more thought into how power-aware coding techniques could be incorporated into a real-world system.

9.4.1 Codes Derived from Lattice Graphs

As shown in Figure 9.1, an XOR-based code constructed may be constructed from a lattice graph. The vertices correspond to parity symbols and the edges correspond to data symbols. In this construction, we construct what we call square lattice graphs; thus, the code must contain exactly p^2 total vertices. The graph consists of p rows $(R_0, R_1, \ldots, R_{p-1})$ and p columns $(C_0, C_1, \ldots, C_{p-1})$. There are exactly 4 parity symbols that have 2 contributing data symbols, $(p-2) \times (p-2)$ parity symbols that have 4 contributing data symbols and $(p-2) \times 4$ parity symbols with 3 contributing data symbols. Such a code will contain exactly $2 \cdot p \cdot (p-1)$ edges (data symbols). An example lattice graph for $p = 4$ and its associated Tanner graph is shown in Figure 9.1.

Assuming a square lattice, the rate of the code can be derived from the number of parity symbols in the lattice. If there are p parity symbols in a row, then there are p^2 total parity symbols. From this we get $2 \cdot (p-1) \cdot p$ data symbols. The rate as p tends to infinity is

$$\begin{aligned}
\lim_{p\to\infty} \frac{2 \cdot (p-1) \cdot p}{p^2 + 2 \cdot (p-1) \cdot p} &= \\
\lim_{p\to\infty} \frac{2 \cdot p^2 - 2 \cdot p}{p^2 + 2 \cdot p^2 - 2 \cdot p} &= \\
\lim_{p\to\infty} \frac{2 \cdot p - 2}{3 \cdot p - 2} &= \frac{2}{3}
\end{aligned}$$

Thus, the rate of this code meets the bound specified by Theorem 8.3.5. In addition, it turns out that k-balanced write group ensembles under an augmented SDCP policy are quite easy to create with this construction. There are two write groups in such an ensemble. The first write group contains all of the devices associated with the rows. The second contains all of the devices associated with the columns.

As an example, consider the code shown in Figure 9.1. If all of the symbols associated with the rows are activated, every active data symbol can be used to service writes. In this case 12 data symbols are used to service writes. Additionally, all 24 data symbols can be reconstructed without activating any additional symbols. The same is true for activating all of the symbols associated with the columns; thus, both write groups are 24-balanced and the entire ensemble is 24-balanced.

9.4.2 Disklets with Disks

If multiple code symbols are mapped to a single device, then whenever the device is active multiple code symbols are available. Such an arrangement can be generated as follows. Assume an erasure code with n total symbols and a set of N disks. Each disk is divided into a set of fixed-sized *disklets*, where each disk has at least two disklets. Suppose n disklets can be created across the N disks in this way. Multiple code symbols can now be mapped to a single device by creating a one-to-one mapping between code symbols and disklets.

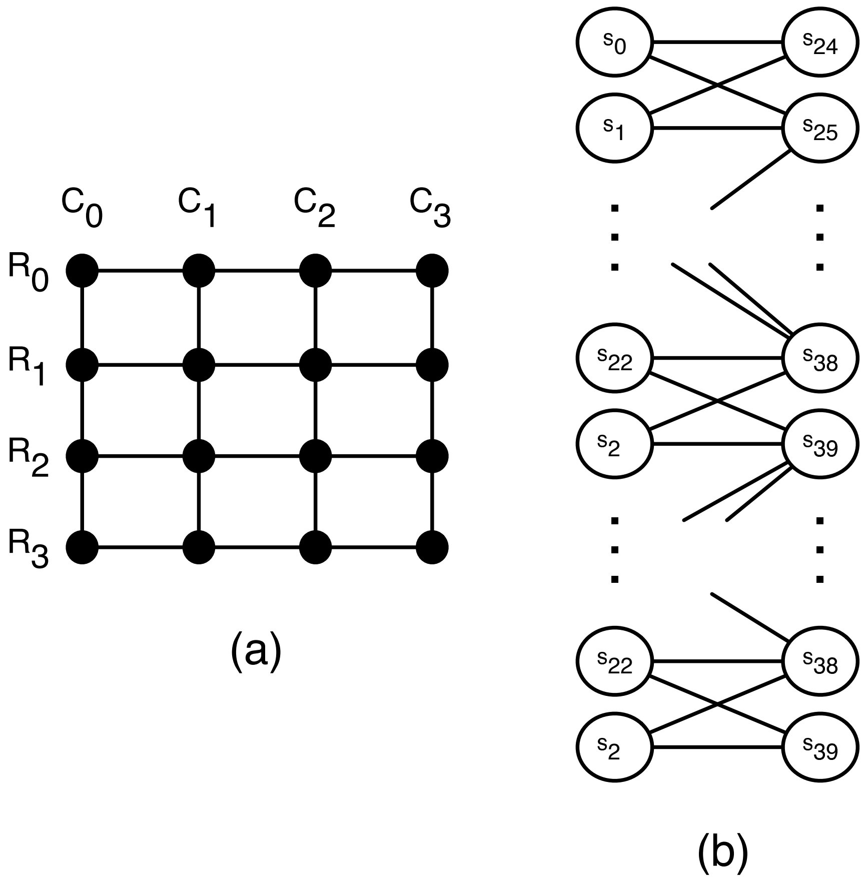


Figure 9.1: Lattice graph.

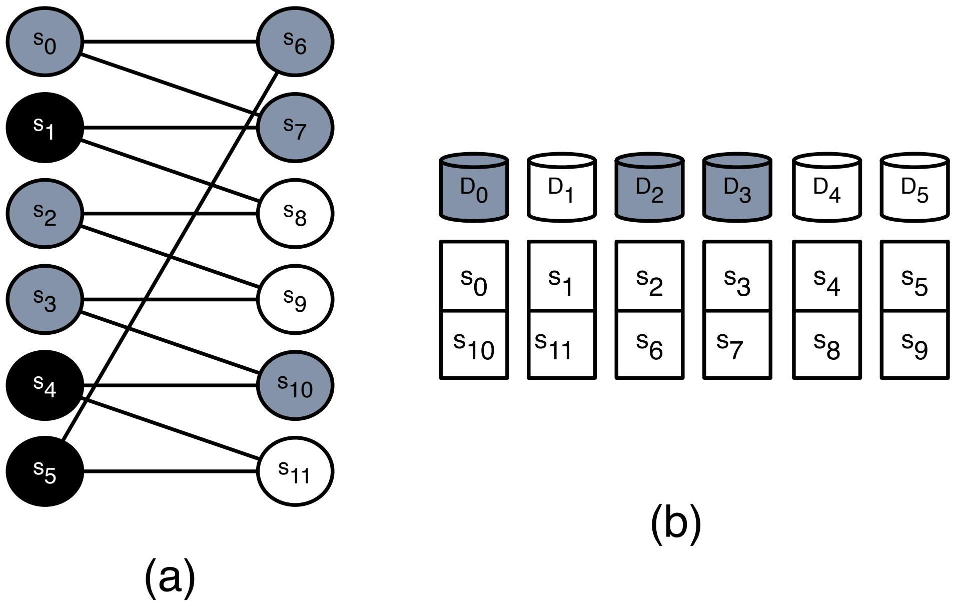


Figure 9.2: Disk with disklets.

Both reliability and expected savings due to power-aware coding will most likely change between mappings, though, there most likely exists isomorphic mappings similar to those shown in Chapter 7. An example erasure code and mapping is shown in Figure 9.2. There are a total of 6 disks, each of which contain 2 disklets. This accommodates all 12 symbols in the code shown Figure 9.2-(a). The mapping shown in Figure 9.2-(b) results in a 2 disk fault-tolerant configuration. Additionally, a 6-balanced write group ensemble can be created.

Under this configuration, the SDCP policy (at the symbol level) results in a 6-balanced ensemble. If the write groups are

$$\{\{D_0, D_2, D_3\}, \{D_1, D_3, D_4\}, \{D_2, D_4, D_5\}, \{D_3, D_0, D_5\}, \{D_4, D_1, D_0\}, \{D_5, D_1, D_2\}\},$$

then a single symbol worth of data can be written and all data symbols are available without additional device activations. The write group $\{D_0, D_2, D_3\}$ is shown in Figure 9.2-(a), where the devices (and symbols) shaded in grey are active and the symbols shaded in black can be reconstructed from the active symbols.

Other types of configurations based on symbol rotation and parity declustering are expected to have similar properties. Such configurations are left to future work.

9.4.3 Disklets with Disk+NVRAM

As shown in the last subsection, the use of disklets can provide additional utility under power-aware coding. Adding non-volatile memory (NVRAM) to the equation can further assist in keeping disks spun down. Currently, the most prevalent form of NVRAM is flash memory. Two attractive properties of flash memory are its low power utilization (compared to disk) and good random read performance. One major issue with current flash memories is endurance: a block can only sustain a certain amount of writes before becoming unreliable. The endurance of a flash memory block can be anywhere from 10^4 writes to 10^6 writes, depending on the technology and level of error-correction implemented on the device's controller. Other NVRAM technologies such as MRAM do not have the same endurance limitations as flash.

Assume that we can create disklets across both disks and flash devices such that each disk has twice the capacity of each flash device. Figure 9.3 shows an example code and fragment mapping across the disklets. Due to the endurance limitation of flash, in most cases it is probably unwise to store parity on flash in a disk-flash hybrid system. Here, we assume that the workload is *write-once, read-maybe*, thus data is rarely re-written. The mapping in Figure 9.3-(b) can tolerate any single device failure. All of the symbols mapped to flash devices can be read or written to without having to spin a device up; thus, their availability is similar to that of an active disk. If D_0 and D_2 are chosen as a write group, then the system can write to the disklets associated with data symbols s_0, s_2, s_4 and s_7. Under this write group, every data symbol is accessible without any additional disk activations.

While this particular architecture may not be feasible in general, it does have some value. Once other NVRAM technologies without the endurance limitation become more widespread, this architecture may be feasible in high-performance systems.

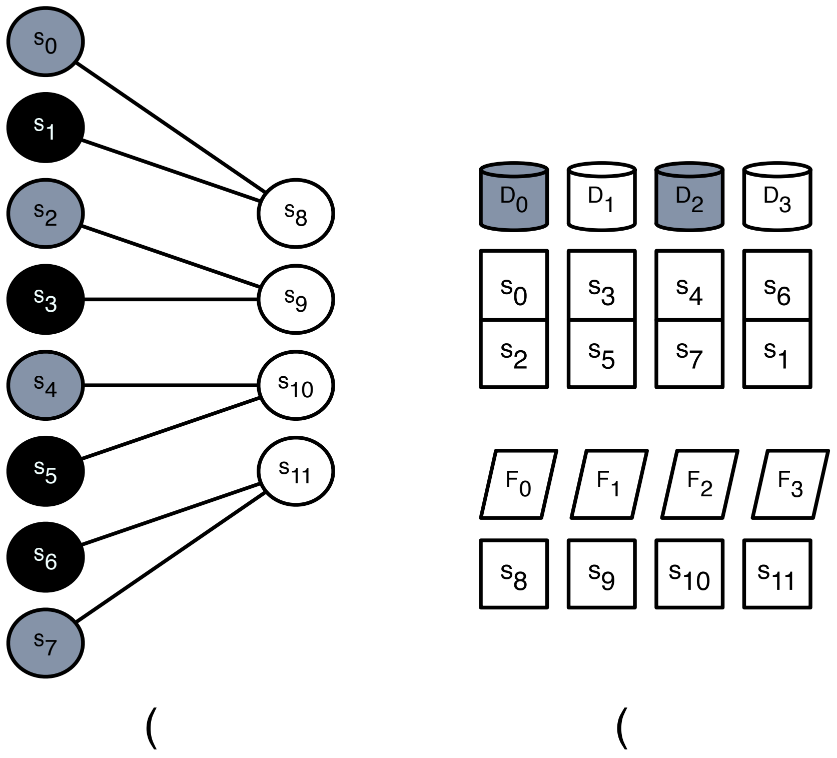
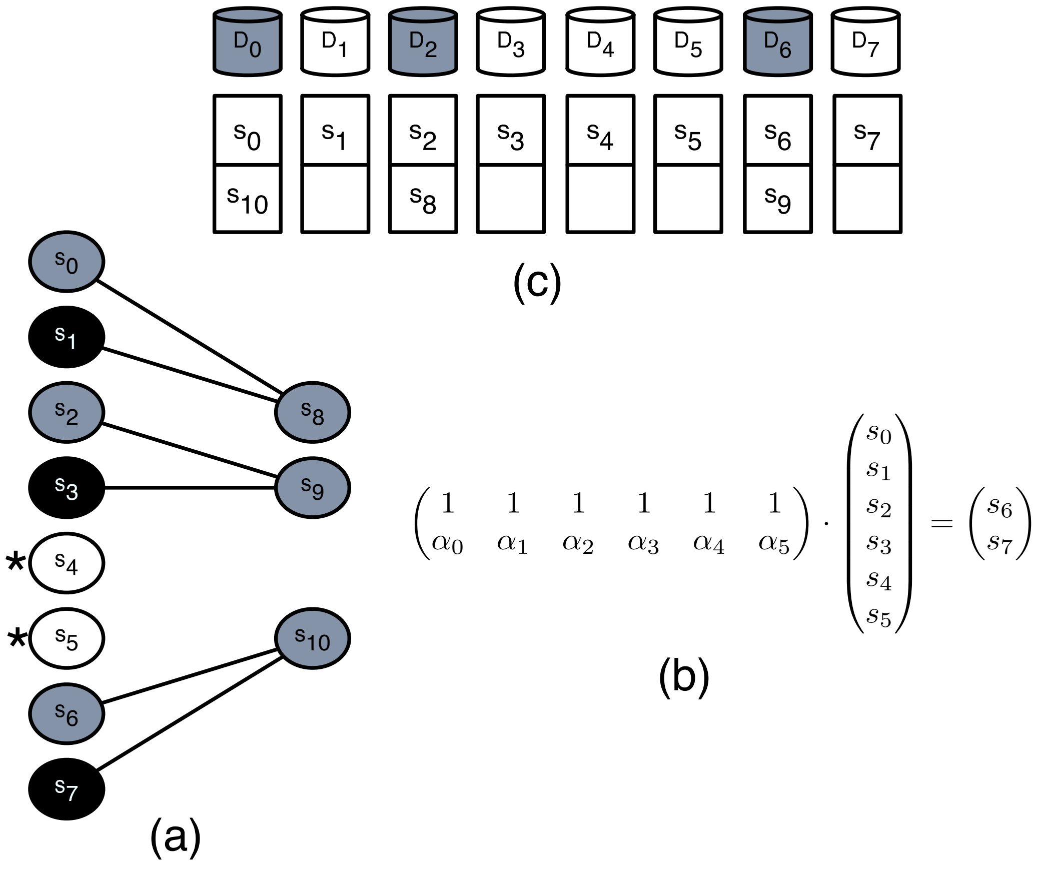


Figure 9.3: Disk+NVRAM



$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ \alpha_0 & \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{pmatrix} = \begin{pmatrix} s_6 \\ s_7 \end{pmatrix}$$

Figure 9.4: Unused space

9.4.4 Utilizing Unused Space on Disk

Finally, consider a generalization of the techniques proposed in PARAID [61]. As in the previous examples, each disk is partitioned into fixed-sized disklets. Instead of creating a one-to-one mapping between disklets and code symbols, the number of disklets exceeds the number of symbols; thus, some disklets remain unmapped. These unmapped disklets can be used to temporarily store parity information for both power savings and temporary fault-tolerance.

Figure 9.4 illustrates an example of using unallocated disklets to keep drives spun-down. The main code used to provide fault-tolerance is a 2 disk fault tolerance MDS code. As shown in the figure, disks D_0 through D_5 are data disks and the remaining disks are devoted to parity. Each disk is partitioned into two disklets: the first disklet is used to create the (6,2)-MDS instance and the second contains unused space.

Suppose the following write group ensemble is created based on an augmented SDCP policy:

$$\{\{D_0, D_2, D_6\}, \{D_1, D_3, D_6\}, \{D_2, D_4, D_7\}, \{D_3, D_5, D_7\}\}.$$

Consider the write group $\{D_0, D_2, D_6\}$. An *impromptu* code instance can be created using a few of the unallocated disklets. Such a code is shown in Figure 9.4-(a). By placing the parity symbols s_8, s_9 and s_{10} on the unallocated disklets of devices D_0, D_2 and D_6, respectively, the write group becomes 6-balanced. In addition, both active data symbols (s_0 and s_2) can absorb writes.

The challenge in using this architecture is migrating data between each write group switch and finding a suitable *impromptu* code to use for each write group. In the example shown in Figure 9.4, the impromptu code can be used for the first two write groups in the ensemble, but will not provide the same properties for the remaining write groups. These challenges are left to future work. We believe that the results of Sections 8.2 and 8.3 can be used to find suitable codes.

Chapter 10

Conclusion

In this thesis we have studied and analyzed the reliability and potential power savings of erasure-coded storage systems. We have identified a variety of issues with current reliability modeling techniques and have created the High-Fidelity Reliability (HFR) Simulator, an efficient, accurate reliability simulation framework for erasure-coded systems.

In Chapter 6, the HFR Simulator was used to perform the most thorough reliability analysis of erasure codes to date. In our analysis, we analyzed the reliability of array codes, flat XOR-based codes and MDS codes. Prior to the HFR Simulator, such an analysis was not possible.

The most common downside to simulation is the possibility of long running times. A version of the HFR Simulator, called HFRS v.2, utilizes a fast simulation technique called importance sampling. Importance sampling allows us to efficiently evaluate the reliability of a highly fault-tolerant system and accurately compare the reliabilities of highly fault-tolerant systems.

The HFR Simulator was used to perform a variety of sensitivity analysis. We showed the effect of sector failure model on overall system reliability. We find that considering the critically exposed region when determining a data loss event can have a dramatic effect on the reliability estimate. Another interesting finding was the sensitivity of SPC codes to increasing sector error rates. Given the symbol-wise Hamming distance of an SPC code, systems that utilize such codes remain robust in the face of sector errors.

The HFR Simulator was used to study the reliability of erasure codes across heterogeneous devices in Chapter 7. We introduced a novel problem in erasure-coded systems, called the *redundancy placement problem*: given a set of heterogeneous devices, how does one map code symbols to devices in a way that maximizes reliability? The Reliability MTTDL estimate was created to quickly order distinct symbol placements and two algorithms were developed in an effort to determine placements that maximize reliability. Both the algorithms and analytic model were validated using the HFR Simulator.

Finally, Chapter 8 introduced the concept of power-aware coding, which relies on the underlying structure of an erasure code to avoid disk activation. The minimal activation policy, called single-data connected-parity, allows both data and parity symbols to be updated without additional disk activation. The most promising result involves erasure codes that exhibit the *balanced* property under the single-data connected-parity policy. We have placed a bound on the rate and provide insight into the structure of such codes. Three metrics are created to explore the power-reliability-space tradeoff. The tradeoff analysis uses the three metrics and the HFR Simulator to estimate the utility of different codes in a system that utilizes power-aware coding.

Bibliography

[1] Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger P. Wattenhofer. Farsite: federated, available, and reliable storage for an incompletely trusted environment. *SIGOPS Oper. Syst. Rev.*, 36(SI):1–14, 2002.

[2] Guillermo A. Alvarez, Walter A. Burkhard, and Flaviu Cristian. Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering. pages 62–72, Denver, CO, June 1997. ACM.

[3] Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler. An analysis of latent sector errors in disk drives. *SIGMETRICS Perform. Eval. Rev.*, 35(1):289–300, 2007.

[4] Mary Baker, Mehul A. Shah, David S. H. Rosenthal, Mema Roussopoulos, Petros Maniatis, TJ Giuli, and Prashanth Bungale. A fresh look at the reliability of long-term digital storage. In *EuroSys-2006: 1st EuroSys Conference*, pages 221–234. ACM, April 2006.

[5] Brett Battles, Cathy Belleville, Susan Grabau, and Judith Maurier. Reducing data center power consumption through efficient storage.

[6] Neerja Bhatnagar, Kevin Greenan, Rosie Wacha, Ethan L. Miller, and Darrell D. E. Long. Energy-reliability trade-offs in sensor networks. June 2008.

[7] Mario Blaum, Jim Brady, Jehoshua Bruck, and Jai Menon. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Trans. Comput.*, 44(2):192–202, 1995.

[8] Walter A. Burkhard and Jai Menon. Disk array storage system reliability. In *Symposium on Fault-Tolerant Computing*, pages 432–441, 1993.

[9] Dennis Colarelli and Dirk Grunwald. Massive arrays of idle disks for storage archives. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[10] Ajay Dholakia, Evangelos Eleftheriou, Xiao-Yu Hu, Ilias Iliadis, Jai Menon, and KK Rao. Analysis of a new intra-disk redundancy scheme for high-reliability raid storage systems in the presence of unrecoverable errors. Technical Report RZ–3652, IBM, 2006.

[11] John R. Douceur and Roger P. Wattenhofer. Optimizing file availability in a secure serverless distributed file system. In *Symposium on Reliable Distributed Systems*. IEEE, 2001.

[12] Richard Durrett. *Essentials of Stochastic Processes*. Springer, 1999.

[13] Jon G. Elerath. A simple equation for estimating reliability of an n+1 redundant array of independent disks (raid). In *International Conference on Dependable Systems and Networks*, 2009.

[14] Jon G. Elerath and Michael Pecht. Enhanced reliability modeling of raid storage systems. In *DSN-2007*, pages 175–184. IEEE, June 2007.

[15] R. Gallager. Low-density parity-check codes. *IEEE Transactions on Information Theory*, 8(1):21–28, Jan 1962.

[16] Abdullah Gharaibeh and Matei Ripeanu. Exploring data reliability tradeoffs in replicated storage systems. In *ACM International Symposium onHigh Performance Distributed Computing (HPDC)*, 2009.

[17] Paul Glasserman, Philip Heidelberger, Perwez Shahabuddin, and Tim Zajic. Splitting for rare event simulation: analysis of simple cases. In *WSC '96: Proceedings of the 28th conference on Winter simulation*, pages 302–308, Washington, DC, USA, 1996. IEEE Computer Society.

[18] Kevin M. Greenan, Ethan L. Miller, and Thomas J. E. Schwarz. Analysis and construction of galois fields for efficient storage reliability. Technical report, University of California, Santa Cruz, 2007.

[19] Kevin M. Greenan, Ethan L. Miller, Thomas J. E. Schwarz, and Darrell D.E. Long. Disaster recovery codes: increasing reliability with large-stripe erasure correcting codes. In *StorageSS '07: Proceedings of the 2007 ACM workshop on Storage security and survivability*, pages 31–36, New York, NY, USA, 2007. ACM.

[20] Kevin M. Greenan, Ethan L. Miller, and Jay J. Wylie. Reliability of xor-based codes on heterogeneous devices. In *The 38th IFIP/IEEE International Conference on Dependable Systems and Networks*, 2008.

[21] James Lee Hafner. Weaver codes : Highly fault tolerant erasure codes for storage systems. In *USENIX Conference on File and Storage Technologies*, 2005.

[22] James Lee Hafner. HoVer erasure codes for disk arrays. In *DSN-2006: The International Conference on Dependable Systems and Networks*, pages 217–226. IEEE, June 2006.

[23] James Lee Hafner, Veera Deenadhayalan, Tapas Kanungo, and KK Rao. Performance metrics for erasure codes in storage systems. Technical Report RJ–10321, IBM, 2004.

[24] James Lee Hafner, Veera Deenadhayalan, KK Rao, and John A. Tomlin. Matrix methods for lost data reconstruction in erasure codes. In *FAST-2005: 4th USENIX Conference on File and Storage Technologies*, pages 183–196. USENIX Association, December 2005.

[25] James Lee Hafner and KK Rao. Notes on reliability models for non-MDS erasure codes. Technical Report RJ–10391, IBM, October 2006.

[26] Danny Harnik, Dalit Naor, and Itai Segall. Low power mode in cloud storage systems. In *IEEE International Symposium on Parallel and Distributed Processing*, 2009.

[27] Mark Holland and Garth Gibson. Parity declustering for continuous operation in redundant disk arrays. In *Architectural Support for Programming Languages and Operating Systems*, pages 23–25, 1992.

[28] Cheng Huang and Lihao Xu. Star: an efficient coding scheme for correcting triple storage node failures. In *FAST'05: Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies*, pages 15–15, Berkeley, CA, USA, 2005. USENIX Association.

[29] Ilias Iliadis, Robert Haas, Xiao-Yu Hu, and Evangelos Eleftheriou. Disk scrubbing versus intra-disk redundancy for high-reliability raid storage systems. In *SIGMETRICS '08: Proceedings of the 2008 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 241–252, New York, NY, USA, 2008. ACM.

[30] Michael G. Thomason James S. Plank. A practical analysis of low-density parity-check erasure codes for wide-area storage applications. In *International Conference on Dependable Systems and Networks*, 2004.

[31] Richard Karp Marek Karpinski Michael Luby David Zuckerman Johannes Blomer, Malik Kalfane. An xor-based erasure-reilient coding scheme. Tech Report, 1995.

[32] S. Kirkpatrick, C.D. Gelatt Jr.., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220, 4598:671–680, May 1983.

[33] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishan Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. Oceanstore: an architecture for global-scale persistent storage. *SIGPLAN Not.*, 35(11):190–201, 2000.

[34] Dong Li and Jun Wang. Eeraid: energy efficient redundant and inexpensive disk array. In *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 29, New York, NY, USA, 2004. ACM.

[35] Mingqiang Li, Jiwu Shu, and Weimin Zheng. GRID codes: Strip-based erasure

codes with high fault tolerance for storage systems. In *ACM Transactions on Storage*, 2009.

[36] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and Their Application.* Cambridge University Press, 1985.

[37] Michael G. Luby, Michael Mitzenmacher, M. Amin Shokrollahi, Daniel A. Spielman, and Volker Stemann. Practical loss-resilient codes. In *STOC 1997: Proceedings of the 29th annual ACM Symposium on Theory of Computing*, pages 150–159. ACM Press, 1997.

[38] Jovan M. Nahman. *Dependability of Engineering Systems.* Springer-Verlag, 2002.

[39] Marvin K. Nakayama and Perwez Shahabuddin. Quick simulation methods for estimating the unreliability of regenerative models of large, highly reliable systems. *Probab. Eng. Inf. Sci.*, 18(3):339–368, 2004.

[40] V.F. Nicola, P. Heidelberger, and P. Shahabuddin. Uniformization and exponential transformation: Techniques for fast simulation of highly dependable non-markovian systems. In *Twenty-Second International Symposium on Fault-Tolerant Computing, 1992.*, pages 130–139, Jul 1992.

[41] V.F. Nicola, M.K. Nakayama, P. Heidelberger, and A. Goyal. Fast simulation of dependability models with general failure, repair and maintenance processes. *Fault-Tolerant Computing, 1990. FTCS-20. Digest of Papers., 20th International Symposium*, pages 491–498, 1990.

[42] V.F. Nicola, P. Shahabuddin, and M.K. Nakayama. Techniques for fast simulation of models of highly dependable systems. *IEEE Transactions on Reliability*, 50(3):246–264, Sep 2001.

[43] G. S. Shedler P. A. W Lewis. Simulation of nonhomogeneous poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413, 1979.

[44] Bernd Panzer-Steindel. Data integrity (draft 1.3), April 2007. CERN/IT.

[45] David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *ACM SIGMOD International Conference on Management of Data*, pages 109–116, June 1988.

[46] Atul Goel Tomislav Grcanac Steven Kleiman James Leong Sunitha Sankar Peter Corbett, Bob English. Row-diagonal parity for double disk failure correction. In *USENIX Conference on File and Storage Technologies*, 2004.

[47] Eduardo Pinheiro and Ricardo Bianchini. Energy conservation techniques for disk array-based servers. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, pages 68–78, New York, NY, USA, 2004. ACM.

[48] Eduardo Pinheiro, Ricardo Bianchini, and Cezary Dubnicki. Exploiting redundancy to conserve energy in storage systems. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 15–26, New York, NY, USA, 2006. ACM.

[49] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. Failure trends in a large disk drive population. In *FAST'07: Proceedings of the 5th conference on USENIX Conference on File and Storage Technologies*, pages 2–2, Berkeley, CA, USA, 2007. USENIX Association.

[50] James S. Plank. Erasure codes for storage applications. Tutorial slides, presented at *FAST-2005: 4th Usenix Conference on File and Storage Technologies*, December 2005.

[51] James S. Plank, Adam L. Buchsbaum, Rebecca L. Collins, and Michael G. Thomason. Small parity-check erasure codes - exploration and observations. In *DSN-2005: The International Conference on Dependable Systems and Networks*, pages 326–335. IEEE, July 2005.

[52] KK Rao, James Lee Hafner, and Richard A. Golding. Reliability for networked storage nodes. In *DSN-2006: The International Conference on Dependable Systems and Networks*, pages 237–248. IEEE, June 2006.

[53] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[54] Yasushi Saito, Svend Frølund, Alistair Veitch, Arif Merchant, and Susan Spence. FAB: Building distributed enterprise disk arrays from commodity components. In *ASPLOS-XI: 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 48–58. ACM Press, October 2004.

[55] Bianca Schroeder and Garth A. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *FAST-2007: 5th USENIX Conference on File and Storage Technologies*, pages 1–16. USENIX Association, 2007.

[56] Thomas J. E. Schwarz, Qin Xin, Ethan L. Miller, Darrell D. E. Long, Andy Hospodor, and Spencer Ng. Disk scrubbing in large archival storage systems. In *MASCOTS '04: Proceedings of the The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)*, pages 409–418, Washington, DC, USA, 2004. IEEE Computer Society.

[57] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, and Kaladhar Voruganti. Potshards: Secure long-term storage without encryption. In *USENIX Annual Technical Conference*, 2007.

[58] Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, and Kaladhar Voruganti. Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage. In *6th USENIX Conference on File and Storage Technologies*, 2008.

[59] Alexander Thomasian and Mario Blaum. Mirrored disk organization reliability analysis. *IEEE Trans. Comput.*, 55(12):1640–1644, 2006.

[60] Jun Wang, Huijun Zhu, and Dong Li. e-RAID: Conserving energy in conventional disk-based RAID system. In *IEEE Transactions on Computers*, 2008.

[61] Charles Weddle, Mathew Oldham, Jin Qian, An-I Andy Wang, Peter Reiher, and

Geoff Kuenning. Paraid: A gear-shifting power-aware raid. *Trans. Storage*, 3(3):13, 2007.

[62] Avani Wildani, Thomas Schwarz, Ethan L. Miller, and Darrell D. E. Long. Protecting against rare event failures in archival systems. September 2009.

[63] Matthew Woitaszek and Henry M. Tufo. Fault tolerance of tornado codes for archival storage. In *15th IEEE International Symposium on High Performance Distributed Computing*, 2006.

[64] Matthew Woitaszek and Henry M. Tufo. Tornado codes for maid archival storage. *msst*, 0:221–226, 2007.

[65] Jay J. Wylie and Ram Swaminathan. Determining fault tolerance of XOR-based erasure codes efficiently. In *DSN-2007*, pages 206–215. IEEE, June 2007.

[66] Qin Xin, Ethan L. Miller, Thomas Schwarz, Darrell D. E. Long, Scott A. Brandt, and Witold Litwin. Reliability mechanisms for very large storage systems. In *MSS '03: Proceedings of the 20 th IEEE/11 th NASA Goddard Conference on Mass Storage Systems and Technologies*, page 146. IEEE Computer Society, 2003.

[67] Qin Xin, J. E. Thomas, S. J. Schwarz, and Ethan L. Miller. Disk infant mortality in large storage systems. In *MASCOTS '05: Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 125–134, Washington, DC, USA, 2005. IEEE Computer Society.

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

[69] Xiaoyu Yao and Jun Wang. Rimac: a novel redundancy-based hierarchical cache architecture for energy efficient, high performance storage systems. *SIGOPS Operating Systems Review*, 40(4):249–262, 2006.

[70] Haifeng Yu, Phillip B. Gibbons, and Suman Nath. Availability of multi-object operations. In *NSDI-2006: Proceedings of the Symposium on Networked Systems Design and Implementation*, May 2006.

[71] Q. Zhu, F. David, C. Devaraj, Z. Li, Y. Zhou, and P. Cao. Reducing energy consumption of disk storage using power-aware cache management. In *The 10th International Symposium on High Performance Computer Architecture*, 2004.

[72] Qingbo Zhu, Zhifeng Chen, Lin Tan, Yuanyuan Zhou, Kimberly Keeton, and John Wilkes. Hibernator: helping disk arrays sleep through the winter. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 177–190, New York, NY, USA, 2005. ACM.

[73] Qingbo Zhu, Asim Shankar, and Yuanyuan Zhou. Pb-lru: a self-tuning power aware storage cache replacement algorithm for conserving disk energy. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, pages 79–88, New York, NY, USA, 2004. ACM.