PRACTICAL ERASURE CODES FOR STORAGE SYSTEMS

THE STUDY OF ENTANGLEMENT CODES, AN APPROACH THAT PROPAGATES REDUNDANCY TO INCREASE RELIABILITY AND PERFORMANCE

Thèse
soutenue le 19 mai 2017
à la Faculté des Sciences
chaire de Systèmes Complexes
programme doctoral
Université de Neuchâtel

pour l'obtention du grade de Docteur ès Sciences
par

Verónica del Carmen Estrada Galiñanes

UNIVERSITÉ DE NEUCHÂTEL

acceptée sur proposition du jury:

Professeur Peter Kropf, président du jury
Université de Neuchâtel, Suisse

Professeur Pascal Felber, directeur de thèse
Université de Neuchâtel, Suisse

Professeur Patrick T. Eugster, rapporteur
TU Darmstadt, Allemagne

Professeur Ethan L. Miller, rapporteur
Université de California à Santa Cruz, États-Unis

Neuchâtel, UNINE, 2017

UNIVERSITÉ DE NEUCHÂTEL

Faculté des Sciences
Secrétariat-décanat de Faculté
Rue Emile-Argand 11
2000 Neuchâtel – Suisse
Tél : + 41 (0)32 718 21 00
E-mail : secretariat.sciences@unine.ch

IMPRIMATUR POUR THESE DE DOCTORAT

La Faculté des sciences de l'Université de Neuchâtel autorise l'impression de la présente thèse soutenue par

Madame Verónica del Carmen Estrada Galiñanes

Titre:

"Practical Erasure Codes for Storage Systems: The Study of Entanglement Codes, an Approach that Propagates Redundancy to Increase Reliability and Performance"

sur le rapport des membres du jury composé comme suit:

- Prof. Pascal Felber, directeur de thèse, UniNE
- Prof. Peter Kropf, UniNE
- Prof. Ethan L. Miller, University of California, Santa Cruz, USA
- Prof. Patrick Eugster, TU Darmstadt, D

Neuchâtel, le 06.06.2017

Le Doyen, Prof. R. Bshary

[illegible]

He explained that an Aleph
is one of the points in space
that contains all other points.
— Jorge Luis Borges (1899-1986),
Argentine writer.

This work
is dedicated to
Vinton Cerf for the Internet,
Brewster Kahle for the Internet Archive,
David S. H. Rosenthal for his blog on digital preservation,
my brother **Alejandro D. V. Estrada** for preserving our family photos,
and **Rudolf E. Martin**, whose love and support sustained me throughout the years . . .

Acknowledgements

First and foremost, I would like to thank my advisor Prof. Pascal Felber for his motivation, patience, dedication and extraordinary support throughout these project. In addition to my advisor, I would like to thank the other members of my thesis committee: Prof. Ethan L. Miller, Prof. Patrick Eugster and Prof. Peter Kropf for their insightful feedback and hard questions.

This thesis would have been impossible without the support of the Swiss National Science Foundation that promotes collaborative research and encourages young researchers. Many thanks to the researchers that make the "Trustworthy Cloud Storage" Sinergia project possible, in particular, from EPFL DIAS Prof. Anastasia Ailamaki, from EPFL DSLAB Prof. George Candea, from ETH Systems Security Group Prof. Srdjan Capkun, from University of Lugano Prof. Fernando Pedone, and from University of Neuchâtel Prof. Pascal Felber. Many thanks to the Department of Computer Science from University of Houston, to the Goldman Research Group from the European Bioinformatics Institute, the ETH Systems Group, and the Department of Mathematics from University of Zürich for inviting me to discussions and as a seminar speaker. Many thanks to all the organizations that make possible my frequent attendance to conferences, workshops and other events: Usenix, Google, ACM-W Europe, ACM, Bureau égalité des chances UNINE, and the Internet Archive. And many thanks to CUSO and the Program REGARD for the organization of a variety of courses, ateliers and winter schools.

I would like to thank the University of Neuchâtel for giving me a wonderful workplace and opening the door to a large network of resources. In addition to a professional network, this period gave me some lifetime friends. Thanks to Dr. Etienne Riviére who was my first contact with the SNSF sinergia project that motivated this thesis and for his kind support to some of my summer projects like GSoC. Thanks to Dr. Hugues Mercier and Dr. Valerio Schiavoni for pushing me with non-conform zone questions. Thanks to Prof. Elisa Gorla, who made me aware that career planning is an important aspect of a research proposal. Thanks to Dr. Relinde Jurrius for her thoughful and pedagogigal comments. She has the ability to listen carefully avoiding preconceived ideas. Thanks to Muriel Besson for her dedication to PhD students in the mentoring program. Thanks to Dr. Anita Sobe, Dr. Patrick Marlier, Andrei Lapin and Rafael Pires for being great teaching assistant partners. Thanks to Dr. José Valerio for his collaboration in the Tahoe prototype that preceded my research for this thesis. Thanks to Maria Carpen-Amarie and Mirco Koher for reviewing some parts of this thesis. Thanks to Emilie Auclair and other secretaries that help me with the administrative work. Thanks to

Yarco, Mascha and to all my colleagues for listening to my rehearsals, reading my unfinished work, sharing with me lunch time, coffee breaks, game nights, via ferratas and a plethora of adventures throughout all these years. Thanks to Raph and to Pascal for introducing me to the Alps and for rescuing me when it was necessary.

I would like to thank the University of California, Santa Cruz, the Jack Baskin School of Engineering, the Center for Research in Storage Systems (CRSS), and the Storage Systems Research Center (SSRC) for hosting my 6-month research doc mobility project. Many thanks to the faculty and associated researchers, in particular to Prof. Ethan L. Miller, Prof. Darrell D. E. Long, Dr. Andy Hospodor, Dr. Ahmed Amer, Dr. Peter Alvaro, Prof. Thomas Schwarz and Prof. Carlos Maltzahn. During my stay, I had the opportunity to attend many seminars, hear interesting talks and discuss with prestigious guests. Many thanks to the participants of the weekly SSRC seminar meetings . Thanks to Cynthia McCarley for her help with administrative issues. Thanks to the International Student and Scholar service office, in particular to Parinaz Zartoshty and Keri Toma for being very kind and helpful towards my inquiries. Thanks to Ivo Gimenez for the nice talk after my PhD crisis and for organizing the social lab events. Thanks to my colleagues, special thanks to Yan Li, Ana McTaggart and Rahul Talari for integrating me in the group and to Andy who donate his house for my farewell party.

I would like to thank many researchers, engineers and friends that volunteer their time asking me plenty of questions, answering my questions, reading my drafts, identifying unclear parts, criticizing my work, but overall caring about my research and consequently, encouraging my writing. Special thanks to my co-author Prof. Jehan-François Pâris, who drove my attention towards simple entanglements. Many thanks to Prof. Ruediger Urbanke, who welcomed me in his office to discuss coding theory. Many thanks to Dr. Kevin Greenan, Dr. Jay J. Wylie, who answered my emails on reliability. Many thanks to Dr. Jay Lofstead for being a good listener. Many thanks to Prof. James S. Plank for all his contributions to the field of erasure coding and for answering my emails regarding the jerasure library. Many thanks to Dr. Luciano Bello and Kragen Javier Sitaker for discussing entanglement codes and the Prolog tool to verify minimal erasure patterns. Many thanks to Théo Beigbeder for help in the translations. Many thanks to Rudolf Martin for several discussions. Many thanks to Dr. Luiz Barroso for responding my enquiries about availability in datacenters. Many thanks to Tim Feldman for answering my emails regarding storage. Many thanks to Dr. Jossy Sayir for his honest advices. Many thanks to Dr. Sage Weil and Ceph engineers for clarifying some aspects of the Ceph system's architecture. Many thanks to my mentors at SIGCOMM 2015, Dr. Paolo Costa and Prof. John Byres. During conferences, poster sessions, doctoral workshops and other events I had the opportunity to discuss different aspects of my research with a large audience, in particular, many thanks, in no particular order, to: Dr. Cheng Huang, Dr. Donald Porter, Dr. Jesus Carretero, Pat Helland, Prof. Christof Fetzer, Prof. Torsten Braun, Prof. Joachim Rosenthal, Dr. Danny Harnik, Dr. Dalit Naor, Dr. Hillel Kolodner, Dr. Eitan Yaakobi, Dr. John Wilkes, Dr. Brent Welch, Prof. Willy Zwaenepoel, Dr. Antonia Wachter-Zeh, Prof. Gustavo Alonso, Dr. Erik Riedel, Robert Krahn, Jonnahtan Saltarin, Prof. Luc Gauthier, Charles Randall and to all my anonymous reviewers.

Many thanks to all the people who are working in projects to preserve digital information. Two projects that raised my attention during my research were the Internet Archive and LOCKSS program based at Stanford Libraries. I'd like to thank the people behind the Internet Archive, in particular, Brewster Kahle, John C. Gonzalez, and Wendy Hanamura who received my visits a couple of times. And thanks to the Chief Scientist of the LOCKSS Program Dr. David S. H. Rosenthal for his constant dedication to his blog about archival storage.

Indirectly, many people prepared me for this PhD journey, in particular I would like to express my deepest gratitute to all my former professors and instructors, in special to Prof. Gabriel Venturino for classes on circuit analysis and to Prof. Ignacio Alvarez-Hamelin for introducing me to complex systems. Special thanks to my master advisor Prof. Aki Nakao, without his trust it would have been impossible to do my master in Japan and my life would be completely different. Many thanks to Dr. Carlos Rosito for his efforts to ensure that my scholarship application would arrived before deadline. Many thanks to Dr. Roberto Perazzo for his encouragement and good advices. Many thanks to Carlos Benitez for his support.

I would like to thank my family and friends whose companionship and support helped me to perservere with my goal. In particular, my special thanks go to my brave mother who gave me the best she could with her scarce resources and to my father for becoming a role model in surviving, a cheerful companion and guide in the recent years. Thanks to my international extended family, many thanks to Lucia san, Iiri san, Gloria san, and Kayo chan for their hospitality and truly kind hearts. Thanks to Graciela, Bibi, Sara, Juan, Romi, Sofi, Citlali, Xochi, Yectli, the whole Correntin troop: tio Rudi, Alice, Roberto, Anita, Erica, Silvia, Caro, Diego and Vero. Thanks to my friends, who accepted my ghost mode while I was surviving heavy loads of work and trips. Thanks to Flor, Regi, Maxi, Marce, Louise, Cèline, Diego, Carla, Eva, Nada, Konstantina, Alfonso, Mami, Ale, Laura, Claudia, Denise, Jorge, Cris, Guil Ovi, Paula, Emi Ovi, Anahi, Fede and Martu for making me feel welcome and like in home wherever I go. Thanks to the women of Japanische Frauenchor of Munich, special thanks to Hiroko, for all the happiness that singing brought to my life and for the warm farewell party. Thanks to my Santa Cruz homemates Kira, John, the "surfer Genevois" Greg, the "tea experter" Paul, Peter "Mowgli", Joelyn, Jim, Brandy, Terry, Megan and her mother, and the HH ghost. Special thanks to Barbara, Caro, and Lean, who were with me in Ezeiza when I left Argentina. Thanks to Sana for understanding my crazy working hours. Thanks to Ian and Cindy for bringing good humor and peaceful energy. Thanks to Mitra for sharing many rides in the carousel and bringing many delightful moments to my life. To conclude, "total thanks" to Rudi for his tremendous generosity and for being my learning partner on the most important things. One of the big things that we experienced as a far apart couple is that during my PhD we travelled the equivalent to 6.5 rounds around the Earth to meet each other.

Neuchâtel, 19 mai 2017 V. E. G.

Preface

This dissertation is submitted for the degree of Doctor of Philosophy at the University of Neuchâtel. The research described herein was conducted under the supervision of Professor Pascal Felber in the Department of Computer Science, University of Neuchâtel, in the periods June 2012 - January 2016 and August 2016 - May 2017. A part of this research was conducted abroad in the Storage Systems Research Center, Jack Baskin School of Engineering, University of California, Santa Cruz, in the period February 2016 - July 2016.

This work is to the best of my knowledge original, except where acknowledgements and references are made to previous work. Neither this, nor any substantially similar dissertation has been or is being submitted for any other degree, diploma or other qualification at any other university.

Neuchâtel, 19 mai 2017 V. E. G.

Abstract

This dissertation deals with the design of practical erasure codes for storage systems. Hardware and logical disk failures are a common source of system failures that may lead to data loss. Nevertheless, it is predicted that spinning disks would remain the standard storage medium in large datacenters. Cloud storage needs efficient codes to become reliable despite its low-cost components. As systems scale in size and complexity, their properties and requirements may change. When data ages, it is usually moved to dedicated archives. Yet the boundaries between storage systems and archives are getting diffuse as we move into applications that require low latency access such as mining data from large scientific archives. Moreover, the centralized approach of cloud backup services brings privacy and economics concerns. Some studies suggest that cooperative peer-to-peer networks are more sustainable for the long term. But peer-to-peer nodes and spinning disks share an undesirable property: both are unreliable. The motivation for this study is to design flexible and practical codes that can provide high fault-tolerance to improve data durability and availability even in catastrophic scenarios.

Survivability comes through the strength built with redundancy. It is difficult to devise a solution based on classic codes that considers all aspects of dependability: availability, reliability, safety, integrity and maintainability. Compromises are generally found through the complex combination of many techniques. This thesis argues that codes that are based exclusively on the use of parallel networks (such as replication) or mainly on the use of serial networks (as it is seen in the split and expand operations behind classic erasure codes) do not leverage all the resources available in a system. Entanglement codes create redundancy by tangling new data blocks with old ones, building entangled data chains that are woven into a growing mesh of interdependent content. We propose: 1) open and close entanglements as more reliable alternatives than mirroring, 2) alpha entanglements to achieve extremely high fault-tolerance with low storage overhead and low repair costs, and 3) spigot codes to reduce the space footprint from entangled data without significant loss of the entanglement's properties. These codes can leverage storage and bandwidth resources efficiently by exploiting the combinatorial power of network reliability. Furthermore, their flexible design based on virtual chains of entangled data yields a scalable and suitable solution to accommodate future requirements. Finally, due to the combinatorial power of entangled data, all in all, dependability is boosted.

Key words: erasure codes, data entanglement, redundancy, replication, combinatorics, network reliability, fault tolerance, survivability, availability, scalable systems, distributed systems, storage systems, long-term storage, archival storage

Résumé

Cette dissertation traite de la conception de codes d'effacement pratiques pour les systèmes de stockage. Les défaillances de disque physiques et logiques sont une source commune d'erreurs ; cependant, il est prédit que les disques dur à plateaux vont rester le support de stockage standard dans les grands centres de données. Le stockage dans le cloud a besoin de codes efficaces pour devenir fiables malgré ses composants à faible coût. À mesure que les systèmes augmentent en taille et en complexité, leurs propriétés et leurs exigences peuvent changer. Lorsque les données vieillissent, elles sont habituellement déplacées vers des archives dédiées. Pourtant, les limites entre les systèmes de stockage et les archives deviennent diffuses à mesure que nous évoluons vers des applications nécessitant un faible temps de latence, telles que l'exploration de données des grandes archives scientifiques. De plus, la centralisation des services de sauvegarde dans le cloud amène des préoccupations en matière de protection de la vie privée et de coûts de maintenance. Certaines études suggèrent que les réseaux coopératifs peer-to-peer sont plus durables à long terme. Mais les noeuds peer-to-peer et les disques dur partagent une propriété indésirable : aucun des deux ne sont assez fiables pour un déploiement à grande échelle La motivation de cette étude est de concevoir un code flexible et pratique qui offre une haute tolérance aux pannes pour améliorer la durabilité et la disponibilité des données même dans des scénarios de type "catastrophe".

La survie des données est basée sur la robustesse et la redondance. Il est difficile de concevoir une solution basée sur des codes classiques qui tiennent compte de ces cinq aspects : disponibilité, fiabilité, sécurité, intégrité et maintenabilité. Les compromis sont généralement obtenus par la combinaison de différentes techniques. Cette thèse cherche à montrer que les techniques de codage basées exclusivement sur l'utilisation de réseaux parallèles (comme la réplication) ou concentrés sur l'utilisation de réseaux en série (comme dans les opérations de division et d'expansion utilisées dans les codes classiques d'effacement) ne permettent pas d'exploiter toutes les ressources disponibles dans le système. Les codes d'enchevêtrement (entanglement codes) créent une redondance en enroulant de nouveaux blocs de données avec les anciens, en construisant des chaînes de données enchevêtrées qui sont tissées dans un maillage croissant de contenu interdépendant. Nous proposons : 1) chaîne enchevêtrements, qui peuvent être ouverts ou fermés, comme des alternatives plus fiables que la mise en miroir de disques, 2) des enchevêtrements alpha pour obtenir une tolérance de panne extrêmement élevée avec un faible coût de stockage et des coûts de réparation faibles, et 3) des codes de robinet (spigot codes) pour réduire l'empreinte spatiale à partir de données enchevêtrées sans perte significative des propriétés de l'enchevêtrement. Ces codes peuvent exploiter efficacement le stockage et les ressources de bande passante en exploitant la puissance combinatoire de la fiabilité du réseau. En outre, leur conception flexible basée sur des chaînes virtuelles de données enchevêtrées offre une solution évolutive et adaptée aux besoins futurs. Enfin, en raison de la puissance combinatoire des données enchevêtrées, dans l'ensemble, les cinq aspects sont renforcés.

Mots clefs : codes d'effacement, enchevêtrement des données, redondance, réplication, combinatoire, fiabilité du réseau, tolérance aux pannes, survie, disponibilité, systèmes évolutifs, systèmes distribués, systèmes de stockage, stockage à long terme, stockage d'archives

Zusammenfassung

Diese Dissertation beschäftigt sich mit der Gestaltung von praktischen Löschcodes für Speichersysteme. Hardware und logische Festplattenfehler sind eine häufige Quelle von Systemfehlern, die zu Datenverlust führen kann. Es wird jedoch vorausgesagt, dass Festplatten das Standard-Speichermedium in grossen Rechenzentren bleiben würden. Cloud-Storage benötigt effiziente Codes, um trotz seiner kostengünstigen Komponenten zuverlässig zu werden. Da Systeme in Grösse und Komplexität wachsen, können sich ihre Eigenschaften und Anforderungen ändern. Veraltete Daten werden normalerweise auf dedizierte Archive transferiert. Die Grenzen zwischen Speichersystemen und Archiven werden immer diffuser mit zunehmendem Einsatz von Anwendungen, die eine kurze Latenzzeit erfordern, wie z.B. Data-Mining aus grossen wissenschaftlichen Archiven. Darüber hinaus bringt der zentralisierte Ansatz von Cloud-Backup-Diensten Bedenken hinsichtlich der Privatsphäre und Wartungskosten auf lange Sicht. Einige Studien deuten darauf hin, dass kooperative Peer-to-Peer-Netzwerke langfristig nachhaltiger sind. Aber Peer-to-Peer-Knoten und Festplatten haben eine unerwünschte Eigenschaft: beide sind unzuverlässig. Die Motivation für diese Studie ist es, flexible und praktische Codes zu entwickeln, die eine hohe Fehlertoleranz zur Verfügung stellen können um die Datenlanglebigkeit und -verfügbarkeit auch in Katastrophenszenarien zu verbessern.

Überlebensfähigkeit eines Speichersystems kommt durch die Stärke, die mit Redundanz gebaut wird. Es ist schwierig, eine Lösung zu entwickeln, die auf klassischen Codes basiert und die die folgenden fünf Aspekte berücksichtigt: Verfügbarkeit, Zuverlässigkeit, Sicherheit, Integrität und Wartbarkeit. Kompromisse werden in der Regel durch die komplexe Kombination von vielen Techniken gefunden. Diese These argumentiert, dass Codierungstechniken, die ausschliesslich auf der Verwendung von parallelen Netzwerken (wie z. B. Replikation) oder hauptsächlich auf der Verwendung von seriellen Netzwerken basieren (wie z. B. Split- und Erweiterungsoperationen in klassischen Löschcodes) nicht alle verfügbaren Ressourcen in einem System nutzen. Verschränkung-Codes (Entanglement-Codes) schaffen Redundanzen durch die Verknüpfung neuer Datenblöcke mit alten, wodurch eine verschränkte Datenkette in ein wachsendes Netz von voneinander abhängigen Inhalten gewebt wird. Wir schlagen vor: 1) offen und geschlossene Verschränkungsketten sind zuverlässigere Alternativen als Spiegelung, 2) Alpha-Verschränkung-Codes, um eine extrem hohe Fehlertoleranz mit niedrigem Speicheraufwand und niedrigen Reparaturkosten zu erzielen, und 3) Zapfencodes (Spigot-Codes), um den Platzbedarf von verschränkten Daten zu reduzieren ohne signifikanten Verlust der Eigenschaften der Verschränkung. Diese Codes können Speicher- und Bandbreitenressourcen effizient nutzen, indem sie die kombinatorische Stärke der Netzwerkzuverlässigkeit nutzen. Darüber hinaus liefert ihr flexibles Design basierend auf virtuellen Ketten von verschränkten Daten eine skalierbare und geeignete Lösung für zukünftige Anforderungen. Schliesslich wird aufgrund der kombinatorische Stärke der verschränkten Daten, alles in allem, die Vertrauenswürdigkeit gesteigert.

Stichwörter: Löschcodes, Datenverschränkung, Redundanz, Replikation, Kombinatorik, Netzwerkzuverlässigkeit, Fehlertoleranz, Überlebensfähigkeit, Verfügbarkeit, skalierbare Systeme, verteilte Systeme, Speichersysteme, Langzeitarchivierung, Archivspeicher

Resumen

Esta tesis trata del diseño de códigos de borrado que sean prácticos para sistemas de almacenamiento de datos. Los errores de disco, sean estos ocasionados por una falla de hardware o una falla lógica, son una fuente común de errores de sistema que pueden ocasionar pérdida de datos. Sin embargo, se predice que los discos duros giratorios seguirían siendo el medio de almacenamiento estándar en grandes centros de datos. El almacenamiento en la nube necesita códigos eficientes para que el sistema sea confiable a pesar de sus componentes de bajo costo. A medida que los sistemas varían en tamaño y complejidad, sus propiedades y requerimientos pueden cambiar. Cuando los datos envejecen, se los suele mover a archivos dedicados. Sin embargo, las fronteras entre los sistemas de almacenamiento y los archivos se vuelven difusas a medida que avanzamos hacia aplicaciones que requieren un acceso de baja latencia, como la minería de datos en grandes archivos científicos. Además, el enfoque centralizado de los servicios de respaldo en la nube trae consigo preocupaciones relacionadas con la privacidad y los costos de mantenimiento a largo plazo. Algunos estudios sugieren que las redes cooperativas peer-to-peer son más sostenibles a largo plazo. Pero los nodos peer-to-peer y los discos giratorios comparten una propiedad indeseable: ambos son poco confiables. La motivación para este estudio es diseñar códigos flexibles y prácticos que pueden proporcionar una alta tolerancia a fallos para mejorar la durabilidad y disponibilidad de datos incluso en escenarios catastróficos.

La supervivencia de los datos se logra con la robustez que ofrece un almacenamiento de datos redundante. Es difícil concebir una solución basada en códigos clásicos que considere estos cinco aspectos: disponibilidad, fiabilidad, seguridad, integridad y mantenibilidad. Los compromisos se encuentran generalmente a través de la compleja combinación de varias técnicas. Esta tesis argumenta que las técnicas de codificación que se basan exclusivamente en el uso de redes paralelas (como la replicación) o principalmente en el uso de redes seriales (como se observa en las operaciones de división y expansión usadas en los códigos de borrado clásicos) no aprovechan todos los recursos disponibles en un sistema. Los códigos de entrelazado de datos (entanglement codes) crean redundancia al mezclar nuevos bloques de datos con los antiguos, construyendo cadenas de datos entrelazados que se tejen en una creciente malla de contenido interdependiente. Proponemos: 1) cadenas de datos entrelazados abiertas y cerradas como alternativas más confiables que el uso de discos espejados, 2) códigos de entrelazados alfa (alpha entanglement codes) para lograr una tolerancia a fallos extremadamente alta con bajos costos de almacenamiento y reparación, y 3) códigos de grifo (spigot codes) para reducir aún mas los gastos de almacenamiento sin pérdida significativa de las propiedades de los códigos de entrelazado. Estos códigos pueden aprovechar los recursos de almacenamiento y ancho de banda eficientemente aprovechando la potencia combinatoria de la confiabilidad de los elementos que conforman la red. Además, su diseño flexible basado en cadenas virtuales de datos entrelazados proporciona una solución escalable y adecuada para adaptarse a los requisitos futuros. Por último, debido a la potencia combinatoria de los datos entrelazados se refuerzan los cinco aspectos arriba mencionados.

Palabras clave: códigos de borrado, entrelazado de datos, datos enredados, redundancia, replicación, combinatoria, confiabilidad de red, tolerancia a fallos, supervivencia, disponibilidad, sistemas escalables, sistemas distribuidos, sistemas de almacenamiento, almacenamiento a largo plazo, archivos

Contents

List of Figures

List of Tables

1 Introduction

The more ambitious plan may have more chances of success [. . .]
provided it is not based on a mere pretension
but on some vision of the things beyond those immediately present.
— George Pólya (1887-1985),
Stanford Professor Emeritus.

The urge to handle big data is boosting all kinds of storage related research. Storage efficiency is a primary concern for the storage community. However, other major issues emerge in the long term. For instance, a retrospective look into the current disparity between technological advances shows that while disk capacity has been growing steadily, the improvements done on input-output operations per second (IOPS) lay behind. Furthermore, recent research efforts towards the design and implementation of dependable systems have increased the awareness of the complex interaction of many factors involved in the survivability of a system. Redundancy methods have a key role in the design of dependable systems. Coding theory is an area of active research with a large variety of open problems to solve.

Current coding techniques are poorly suited to consider all aspects of dependability: availability, reliability, safety, integrity and maintenability. As a result, systems grow in complexity due to the need of combininig multiple subsystems or layers to address different problems. There is a considerable amount of literature on codes that optimally satisfy the industry de-facto two-three failures fault tolerance, but very little is known about practical codes that go beyond the standard. A storage system that guarantees high reliability with a low replication factor, i.e., low fault tolerance, must have important maintenance cost. Even if that economic equation works for the cloud storage industry, other sectors would benefit from codes that provide higher fault tolerance efficiently. A growing body of literature has examined the bandwidth overhead caused by coding. Much is known about optimal trade-offs between storage and repair bandwidth, yet little is known about codes with high fault tolerance since research has tended to focus on codes that cause little storage overhead (typically less than two times storage). That restriction causes a partial understanding of the efficiency in a wide range of code settings. Recent work mostly proposes simple optimizations based on classical codes. One example is the design of codes that improve locality [1] based on Reed Solomon codes. Finally, codes are being designed, implemented, and re-adapted by researchers from various fields. Domain specific languages used by mathematicians, cryptographers, graph theoreticians, engineers and computer scientists may become a barrier to understanding.

In spite of the challenges mentioned above, humans and machines will keep generating a deluge of raw and processed data. Many storage applications reached the exabyte scale, for example scientific applications at CERN (Peters and Janyst, 2011), DNA storage (Goldman et al., 2013) and exabyte radio astronomy (Vermij et al., 2014), meaning that it is wise to rethink the technology that supports those systems. In addition, it becomes important to investigate solutions that go beyond short-term goals, to consider data preservation and discuss how to enable data sharing repositories in the long-term. Big data has a positive impact in the generation of knowledge. Access to data improves our life by enhancing our interactions in diverse domains such as science, business, and democracy. The potential of sharing data can be seen on multiple tangible cases. For example, on the Franco-Swiss border, the experiments conducted with the Large Hadron Collinder at CERN generate tens of PB per year. CERN's data is used by a global community of scientists. The ultimate aim of data preservation is to share data with future generations.

Redundancy schemes are necessary to provide secure and long-term data storage. Most existing storage systems store data redundantly using replication, RAID or erasure coding. Although these solutions are effective, the sole use of redundancy does not guarantee all the aspects of dependability. During the design and implementation of a system, a large number of compromises are taken in order to find the right balance between storage cost, rebuild time, and repair bandwidth. Hence, the success of current methods stands on the existence of continuous monitoring and repair operations to keep data safe.

My main argument is that to provide high availability and high faul tolerance with an efficient use of storage and bandwidth resources in a decentralized or centralized storage system we need to go beyond the current models of redundancy. If we dig deeper into the foundations of information theory, we find that the generation of redundancy in classical coding methods is largely based on a model that considers the statistics in communications that move data from source to destination, but ignores the failure model and complexities of large scale distributed systems.

The aim of this thesis is to provide a model to store data redundantly based on data entanglements and to examine the benefits and limitations of this new model in the context of storage applications. Entanglement codes create redundancy by tangling new data blocks with old ones, creating chains of entangled data that are combined into a mesh. The elements

[1] Locality is a property that defines the cost to repair a single failure.

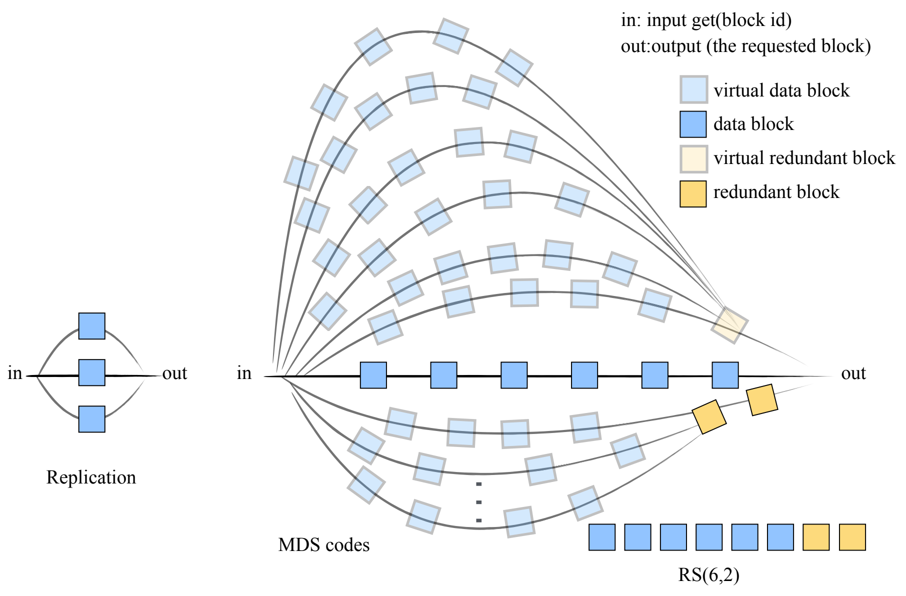


Figure 1.1 – Classical redundancy methods for storage systems. Replication exploits the basic notion of parallelism and replicas, while storage optimal MDS codes such Reed-Solomon codes expand the information with redundant blocks and resourcefully combine them creating multiple paths made of "virtual" blocks.

that form this mesh can be mapped to storage locations in a one-to-one or a many-to-many relationship, in such a way that redundancy is continuously propagated across the system as new information is ingested by the system. The coding technique mechanism used for data entanglement is lightweight and efficient, essentially based on exclusive-or operations.

A major innovation is how entanglement codes weave a virtual layer of interdependent content (and storage devices) that scales with the size of the system. Dependable systems will benefit from multiple paths to access and recover data, efficient repair even in the event of catastrophic failures, and data integrity since the growing layer of interconnected storage elements is an inherent tamper deterrent mechanism, which additionally makes it possible to audit the content. Both models are illustrated in Figure 1.1 and Figure 1.2 respectively. In engineering, the redundancy of a system is explained in terms of how the components are arranged in serial and parallel combinations. If elements are arranged in series, it means that the system needs all elements to satisfactorily perform the task. Elements that are arranged in parallel increase the chances that the system will operate successfully. At least one path that connects input to output is necessary to read data in the figures.

In Figure 1.1 replication exploits the basic notion of parallelism and replicas, while MDS codes such as Reed-Solomon codes expand the information with redundant blocks and resourcefully combine them by creating multiple paths made of "virtual" blocks. The replication example has three replicas of the same document and the system can successfully answer requests if at least one replica is available. The erasure coding example shows a setting RS(6,2) that

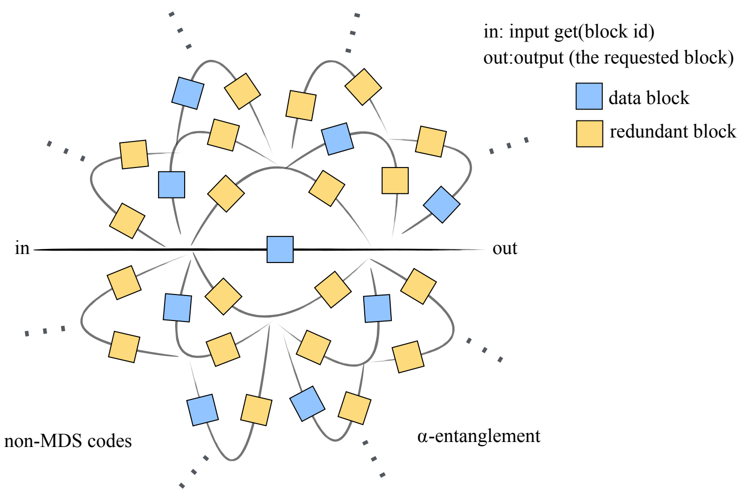


Figure 1.2 – Redundancy through data entanglement. The process creates concentric paths for each data block. The paths that are close to the centre have less elements in serial combinations while the paths that are more distant require more blocks but increase the chances of success in catastrophic failures.

divides a document in six data blocks and expands them with two more blocks. The underlying mathematics of the coding method make possible to recover the document with any six blocks. For that reason, the figure shows many possible paths formed with a serial combination of six blocks. The so-called virtual blocks do not need to be physically replicated, although they belong to more than one path. The combinatorial power of classical codes gives the system more chances to satisfactorily respond a read request. Both options, three-way replication and the erasure coding example, tolerate two failures, but erasure coding requires 1.33x storage while replication requires 3x storage. However, there are two main problems with classical coding: 1) each time a single failure occurs the decoder needs a prohibitive amount of bandwidth (6x overhead for the example provided), and 2) erasure coding techniques are optimal when the code is long, which is typical in digital transmissions but not in storage applications. The network bandwidth that is consumed during single-failure repairs in classical codes like in the example is usually not acceptable for storage applications.

In Figure 1.2 we observe a block-centric view of the virtual layer created by entanglements. The example was generated with a setting $\alpha = 2$, which means that each data block will generate two additional encoded blocks. The process of entangling data creates concentric paths for each data block. The paths that are close to the centre have less elements in serial combinations while the paths that are more distant require more blocks but increase the chances of success in catastrophic failures. Since the paths are interconnected, there are multiple possible combinations to repair data. More details about the principle of entanglements will be given in the thesis.

1.1 Contributions

Untangling code approaches in the context of storage systems. The background and related work to this thesis comprise a body of approaches originated by groups motivated by different aims: some seeking answers to theoretical questions and some trying to address practical concerns. We provide a survey to understand the general panorama of coding and storage systems. It contains the results of studies on a large number of coding approaches, storage systems, and discussions with experts in the field. This review seeks to reconcile theory and practice using approaches from information theory, cryptography and storage communities. The first section guides the reader into information theory concepts to pave the way for understanding erasure coding approaches. The second section goes into more detail on coding techniques that became mainstream for storage applications and other codes that contributed substantially to our understanding. The third section gives the big picture of the storage sector and provides an account of the challenges and open problems in the area. The final section covers data entanglement.

Simple Entanglement Codes. Despite recent advances in solid state storage, the lower cost of magnetic disk drives causes them to remain today the most widespread storage medium in large data centers. One of the main disadvantage of these drives is their poor reliability. We propose simple entanglements as an alternative to mirroring for log structured append-only storage systems. We investigate open and closed entanglements, two simple data distribution layouts for disk arrays. Both techniques use equal numbers of data and parity drives and generate their parity data by computing the exclusive-or (XOR) of the most recently appended data with the contents of their last parity drive. They have the same space overhead as mirroring. While open entanglements maintain an open chain of data and parity drives, closed entanglements include the exclusive-or of the contents of their first and last data drives. We evaluate five year reliabilities of open and closed entanglements, for two different array sizes and drive failure rates. Our results show that open entanglements provide much better five year reliabilities than mirroring, and reduce the probability of data loss by at least 90 percent over a period of five years. Closed entanglements perform even better and reduce the probability of data loss by at least 98 percent.

Alpha Entanglement Codes. We introduce *alpha entanglement codes*, a mechanism to propagate redundant information across a large scale distributed system and create a virtual storage layer of highly interconnected elements. Alpha entanglement codes generalize the concepts explored in simple and helical entanglement codes, our prior work on data entanglement. A central idea in the coding process is the creation of entanglement chains that alternate two elements: data blocks and parity blocks. Parity blocks contain redundant information from previous elements in the chain. Thus, the chain reveals temporal relationship between blocks since the most recent blocks are always written at the head. The encoder builds chains by computing the XOR of two consecutive blocks at the head of a chain and inserts the output adjacent to the last block. Since unencoded data blocks are stored in the system there is no need for decoding in the absence of failures. The code has three parameters. The parameter α

increases storage overhead linearly (by defining the number of chains in which the data block participates) but increases the possible paths to recover data exponentially. The other two parameters increase fault tolerance even further without the need of additional storage by controlling the vicinity of the data blocks.

Entanglement codes have a strong positive impact on data durability due to their ability to repair efficiently single and catastrophic failures. This novel approach can be used to design a robust redundancy scheme for a survivable storage system that would provide high availability, durability and integrity despite external and internal threats with little maintenance. We present the model, the encoder and decoder algorithms, analysis and simulations to understand the code behaviour and compare them with other codes in practical settings.

Unlike other codes, entanglements take advantage of the combinatorial effect while being less dependent on the availability of nodes in a distributed systems. Classical coding is more efficient than replication due to the combinatorial effect. However, in peer-to-peer systems, the negative effects of poor peer availability affects classical coding since the repair process depends on the availability of multiple locations. Alpha entanglements are less affected by the peer availability effect since only two blocks are used for repairs and they benefit from the combinatorial effect too due to the redundancy propagation.

Reliability and Fault Tolerance. Due to the complex relationships built by entanglement codes, the analysis of fault tolerance and reliability is far from being a trivial task. We study this topic based on the properties of the graph that represents the relationships between the entangled data. The main goal of this study is to understand the impact that code settings variations have on fault tolerance. The code parameters regulate the shape of the entanglement graph, therefore, they have an effect on the way redundant data is propagated. The graph is resilient to a large set of erasure patterns, however some data loss is possible when certain patterns appear. Furthermore, the code is irregular, meaning that it can tolerate some patterns of a certain size but not all of them. We analyze erasure patterns and present upper and lower bounds for the data propagation method. To further complete the assessment we present a reliability study based on combinatorial methods to have a comparison measurement with other codes in a theoretical context.

Spigot Codes. Spigot codes are built on top of entanglement codes to tackle a limitation in the way that alpha entanglement codes grow in storage overhead. The storage efficiency decreases with a factor of $1/(\alpha+1)$, where α is the number of chains in which a data block participates. We investigate the suitability of different techniques to increase the efficiency of entanglement codes. The outcome of this study are spigot codes, a solution to increase the code rate and reduce the space footprint from entangled data without significant loss of entanglement's properties. We study two techniques based on code puncturing and XOR coding and present four spigot codes.

1.2 Thesis Organization

The roadmap for this dissertation is as follows:

Chapter 2 provides the general panorama of the past-to-present highlights of information theory, the storage sector, and the research community with the aim to clarify misunderstandings. We discuss challenges and open problems in the storage sector.

Chapter 3 presents simple entanglement layouts for disk arrays. Notwithstanding its simple approach to do entanglements, the study of five year reliability shows that entangled arrays outperform mirroring arrays. This chapter is largely based on our publication in IPCCC'16 proceedings (Estrada-Galinanes et al., 2016).

Chapter 4 introduces complex entanglements comprised by our initial work on helical entanglement codes and its more advanced form: alpha entanglement codes. This chapter presents the entanglement model, the encoder and decoder algorithms and evaluations done in a simulated environment. Helical entanglement codes was published in SSS'13 (Estrada-Galinanes and Felber, 2013), our recent work alpha entanglement codes was submitted for publication (Estrada-Galinanes et al., 2017b)and the research framework to do simulations will appear as extended abstract in SYSTOR'17 (Estrada-Galinanes, 2017).

Chapter 5 details the patterns that define the fault tolerance of entanglement codes and presents upper and lower bounds for the data propagation method. To further complete the assessment, we present a reliability study based on combinatorial methods. The first part of this chapter delves deeper into the study of alpha entanglements and the second part extends the work publised in CLUSTER'15 (Estrada-Galinanes and Felber, 2015b).

Chapter 6 presents spigot codes, a solution based on entanglement codes to improve the storage efficiency without significant loss of entanglement's properties. This chapter is largely based on recent submitted work (Estrada-Galinanes et al., 2017a).

Chapter 7 presents a guideline for implementing entanglement codes. Alpha entanglement codes are not restricted to a particular storage architecture. In particular we present two use cases: the first case is a cooperative storage system built on top of a decentralized database, the second example showcases a centralized storage architecture based on disk arrays. This chapter discussed topics related with the implementation of entanglement codes based on ideas presented in FAST'15 and '16 Work in Progress Sessions talks (Estrada-Galinanes and Felber, 2015a; Estrada-Galinanes and Felber, 2016), Eurosys Doctoral Workshop (Estrada-Galinanes, 2016) and technical reports for my doctoral program (Estrada Galinanes, 2013; Estrada Galinanes, 2015).

Chapter 8 concludes the study by providing a summary of the presented material of this dissertation, discusses its significance and prospects for different communities, and elaborates on future research directions.

2 Background and Related Work

This chapter gives the general panorama of the past-to-present highlights of information theory, the storage sector, and the research community. The main goal is to to fill holes in the literature and clarify commonalities and differences between research conducted by different communities and our work. We strive to provide a historical and theoretical perspective to frame the evolution of redundancy methods while establishing connections with our work. We do not pretend to cover all topics in detail to avoid making the chapter too dense. Instead, we focus on the most relevant work and outline significant contributions and limitations of the approaches taken by other authors. Additionally, we discuss data entanglement and the state of the art at the time we took this line of research in 2012. Other authors have been actively studied the effects of tangling (mixing) data for network coding, as well as for storage systems. Later chapters expand on many of the concepts presented here and introduce additional work.

2.1 Introduction

Last year would have been Claude Shannon 100th birthday. Before Shannon, communication was strictly an engineering discipline. Some of the most well-known codes of that time were Morse and Pulse Code Modulation. In general, standard techniques were the result of heuristic methods and “black art” without a unifying theory.

Shannon’s landmark paper (Shannon, 1948) describes the main components of a communication system and provides a method to quantify information. Shannon developed the foundations for compression (source coding) and digital communications (channel coding). He established a framework for optimal transmission in the presence of noise. Due to his contributions, he is considered the father of information theory. His legacy has a continuous impact in modern communications. Information theory provides the elementary notions for data encoding methods used in current systems.

In the 70s, part of the community believed that “coding is dead”. In an infamous workshop talk, coding theorists were compared with rats abandoned in a confined space deprived from reality for a long time (Lucky, 1993). The forecast was wrong, coding theory was not dead. Though, since then, there was a significant change in the techniques to design new codes.

Algebraic coding was found inadequate to design codes that approach the Shannon's limit. Instead, random coding or "probabilistic coding" became popular since almost every code with potentially infinite blocks can approach the Shannon's limit. Random codes are good but the decoding complexity requires exponential resources (MacKay, 2003). Consequently, research focus was redirected towards codes that have an structure to facilitate implementation, particularly to reduce encoding and decoding complexity. Further developments include graph theory to construct codes. Acyclic Tanner graphs (Tanner, 1981) are extremely efficient for decoding, albeit with suboptimal performance.

In its beginning, the advances of information theory were largely focused on the fundamental problem of reliable communications. The invention of the IBM hard disk and later storage developments (Goddard and Lynott, 1970) expanded the use of coding theory. As David MacKay explained, the communication channel is seen as an abstraction that does not necessarily mean moving bits from one location to another, for example, it could refer to a hard disk (MacKay, 2003). Coding for storage systems became an active research area.

2.2 Basic Concepts of Information Theory

There are different approaches to design codes. This section details some notions and definitions of information theory and, when adequate, their connection with the concepts discussed in this thesis.

Algebraic coding is an approach to design codes based on *linear algebraic equations*. The use of algebraic coding is on decline. Other common approaches are:

- the *generator matrix* that describes how to convert the source bits into the transmitted bits;
- the *parity check matrix* that states the constraints associated to a valid codeword; and
- the *trellis diagram* that specifies the valid codewords with paths in a graph.

These approaches are used selectively, for example, the trellis diagram is useful for **linear block codes** and **convolutional codes**. In a few words, the main differences between linear block codes and convolutional codes is that the first encode data blocks without memory, while the second encode bits and the encoding depends on previous states.

How are they related with the entanglement codes introduced in this thesis? The output of entanglement codes is obtained by encoding the input with previously encoded data that could be retrieved from different and remote storage locations or obtained from the state of the encoder register. Entanglement codes can encode a stream of bits as well as blocks, but we only known a practical implementation for block encoding/decoding. Besides, large blocks make more sense for storage systems. Therefore, the encoding and decoding methods explained in this thesis use data blocks, except for the example of the practical hardware encoder presented in Section 4.4 that explicitly shows how bit encoding works.

2.2.1 Trellis Diagram vs. Helical Lattice

A trellis diagram is a well-known graph used for decoding convolutional codes. In Chapter 4, we define a helical lattice to model the redundant propagation and the possible paths to recover a data chunk from the storage system. Information theoreticians may found superficial similarities with the trellis diagram. In spite of the apparent similarities, a helical lattice should not be confused with a trellis diagram. The lattice diagram describes a storage overlay that shows the dependencies between storage devices/nodes. In terms of information theory, a storage system is an unconstrained channel in which new devices/nodes can appear at any time. The lattice length is variable and the elements of the lattice (nodes and edges) can be mapped to storage locations in a one-to-one or a many-to-many relationship. Most important, a trellis is used for efficient maximum-likelihood decoding while decoding for entanglement codes works in a very different way.

2.2.2 Error Correcting Codes vs. Erasure Codes

A characteristic of storage systems is that the failure location is generally known. [1] Hence, the decoder knows exactly which information is missing. In entanglement codes, the faulty storage location can be mapped to the lattice and repair algorithms can make use of the information. Codes that make use of this information are called erasure codes. Conversely, the location of errors is not known in typical digital communications when bits are transmitted from source to destination through a noisy channel. Codes that decode data without knowing the position of the error are called error correcting codes. Typically, error correcting codes need twice as many redundant symbols as erasure codes to correct a single error.

2.2.3 General Notions of Fault-Tolerance

We give some notions to determine the fault tolerance of a linear code with parameters (n, k), where k are information symbols and $n-k$ are redundant symbols obtained using linear algebraic equations. Perhaps, the first step in designing an error-correcting code is to verify the number of error patterns and the number of possible syndromes enabled by the code equations. The *syndrome*, the *minimum distance* and the *weight enumerator* are fundamental concepts to characterize the fault tolerance of a code.

A syndrome is the pattern of violations for the parity checks rules (MacKay, 2003), i.e., the

[1] Storage systems usually monitor the status of nodes/storage devices.

read syndrome explains a certain flipped bit pattern that does not correspond to codeword. A syndrome zero means that all parity check rules are satisfied. A t-error-correcting code must provide a distinct syndrome for all possible distinguished error patterns of weight up to t in the $N = n$ transmitted bits. Borrowing the explanations and example provided in MacKay book, the syndrome for a (n, k) code is a list of M bits, hence, the maximum possible number of syndromes is 2^M with $M = n - k$, and the number of possible patterns of weight up to *two* is the sum of all combinations of 2-size patterns, 1-size patterns and 0-size patterns as in

$$\binom{N}{2} + \binom{N}{1} + \binom{N}{0}$$

To illustrate with numbers, a $(14, 8)$ code has $2^6 = 64$ distinct syndromes but the possible patterns of weight up to $t = 2$ is 106. Therefore, it is not a 2-error-correcting code.

The minimum distance of a code, also denoted as *Hamming distance* d, is an important property related to the previous concept. It tells the smallest separation between two of its codewords u and v. Let us say that $u = 1010010$ is the transmitted word for source bits 1010, that $v = 1011001$ is the transmitted word for the source bits 1011 and the distance $d = 3$ says that $u - v = 0000111$ has three non-zero elements. The distance also tells us the number of substitutions that are required to convert the codeword u in v. As a rule, the number of errors that the code guarantees is $t = \lfloor (d-1)/2 \rfloor$. Finding the minimum distance of a linear code is NP hard (Vardy, 1997).

In coding theory, the weight enumerator of a code $A(w)$, known as weight or distance distribution, is used as a reference to express the probability of correct decoding, decoding error and decoding failure. It is the number of codewords that have weight w, where weight is the number of “1”s in a codeword.

2.2.4 Channel Capacity

Shannon was interested in measuring code’s performance. He solved a fundamental problem for achieving efficient communications. The channel capacity, also known as *Shannon’s limit*, is the maximum (non-zero) rate $R = C$ at which information can be transmitted through a given channel with an arbitrary small probability of error p_b. The Shannon’s limit tell us how much redundant information is needed to transmit data through a noisy channel. When Shannon’s noisy-channel coding theorem is applied to storage systems, it can determine how many disk drives are needed to store data reliable, e.g., with $p_b = 10^{-15}$.

2.2.5 MDS vs. non-MDS Codes

A linear (n, k, d) code that takes as input words of length k and generates codewords of length n is said to be a maximum distance separable (MDS) code if $d = n - k + 1$, i.e., every k columns of the generator matrix are linearly independent. Maximum distance separable (MDS) codes can tolerate all erasures that are smaller than the Hamming distance. In the context of storage systems, the codeword length n is usually dropped. Codes are denoted by (k, m)-MDS where $m = n - k$ is the number of arbitrary failures that the code can tolerate. When a file or any data object is encoded, the parameter k refers to the number of blocks that are created. In other words, the encoding of a file requires to split it in k blocks and expand it with m extra blocks. The file is decoded with any k blocks out of the total $k + m$ blocks. It is worth noticing that data replication can be expressed using the same notation with k equals to one and m equals to the number of replicas.

Measuring reliability in systems using non-MDS erasure codes is more difficult because the codes do not behave regularly. This irregular fault-tolerance means that the decoder can restore data from some erasure patterns but fails with other patterns, in spite of having the same size. Greenan and co-authors observed that understanding those patterns is of particular relevance to maximize reliability when mapping encoded data to storage devices (Greenan et al., 2008a).

2.2.6 Practical Erasure Codes

A widely accepted definition of practical codes was given by MacKay:
Practical codes are code families that can be encoded and decoded in time and space polynomial in the blocklength (MacKay, 2003).

Shannon's theorem requires the blocklength to be large. A caveat is that the majority of code implementations require computations that are exponential in the blocklength.

The definition is precise enough to facilitate comparisons between codes. Yet, we would like to expand the meaning of practical codes to include our vision for long-term storage applications. In this thesis, the term "practical erasure codes" conveys three major aspects of this work: 1) the efficient use of storage and bandwidth resources to encode and decode data; 2) the flexibility to accommodate future requirements; and 3) an effort to bridge the gap between theory and practice (many codes are good in theory, but in practice only some configurations—sometimes requiring "magic numbers"—work).

2.2.7 Capacity-approaching Codes

Low-density parity check (LDPC) codes (Gallager, 1962) are linear. LDPC codes were invented by Gallager in the 60's but neglected for many decades as they were judged impractical to implement, even though, now they are widely used (Nagaraj, 2017). Because they are described by a low density graph, their generator matrix is sparse but not systematic (defined later in this chapter). Sparse matrix computations are faster than dense matrix, e.g., Reed-Solomon codes (presented in the next section), however the generation of this matrix is expensive. Techniques for efficient encoding appeared later (MacKay et al., 1999; Richardson and Urbanke, 2001). A recent study on their applications in magnetic drives and memory storage systems was done by Jeon (Jeon, 2010). Authors presented irregular LDPC codes that perform very close to Shannon's limit (Richardson et al., 2001).

A problem indicated by Plank and Thomanson is that LDPC literature is "heavy in theory but light in practice" (Plank and Thomason, 2003). Their study addressed five interesting questions all related with the performance of LDPC codes in practical implementations considering finite systems for smaller n. They verified that LDPC codes behave asymptotically and stated that finding good LDPC codes with smaller n is a "black art". They tried to elucidate trade-offs between Reed-Solomon codes and LDPC codes and concluded that there are practical situations in which the first outperforms the second and vice versa. Finally, they raised the issue of patent infringement for experimental use of patented codes since an exception that protects researchers was recently questioned at court.

Parallel concatenated convolutional codes (Benedetto and Montorsi, 1996) have some similarities with our work. Convolutional codes depart from the linear block codes discussed previously. The input sequence is combined using simple additions in GF(2), i.e., exclusive-or (XOR) operations. The codewords have no predefined length but the codes have a constraint length. The encoder processes the source input in a series of k memory registers and shifts the state of one register to the next one at each clock cycle. The n outputs are XOR combinations of different memory registers. Multiple linear-feedback shift-registers can be used to generate convolutional codes. The constraint length K indicates the code's depth, which corresponds to the maximum number of states of the encoder (memory registers). Convolutional codes are described with a Trellis diagram.

Turbo codes (Berrou et al., 1993) are high-performance capacity-approaching codes. They are composed by various elements: convolutional codes, commonly two, and interleavers. The interleaver applies a permutation to the input before it is fed into a convolutional code. A common example is constituted by a systematic encoder, and two convolutional codes connected in parallel using interleavers and has rate $\frac{1}{3}$. *Puncturing* is a technique to achieve higher code rates by eliminating periodically some output parity bits. It has been reported that puncturing reduces the code performance (Chatzigeorgiou et al., 2009).

The *Viterbi algorithm* (Viterbi, 1967) is one of the most well-known decoder for convolutional and turbo codes. It seeks the path in the trellis with minimal cost, i.e., the path formed with the most probable state sequence. As indicated by other authors, the running time of Viterbi grows exponentially with the constraint length of the code. To solve this problem, they proposed the *lazy Viterbi algorithm* which under reasonable noisy conditions has a run time independent of the constraint length (Feldman et al., 2002). None of this decoding algorithms are applicable to entanglement codes that are designed to locally decode individual blocks even if the system distributes encoded blocks in geographically dispersed storage nodes.

2.3 Codes for Storage Applications

In this section, we focus on coding techniques that became mainstream for storage applications and other codes that contributed substantially to our understanding. Many distributed storage systems split files into fixed size blocks, encode them with erasure codes, and store all blocks on potentially geographically distributed servers (Kubiatowicz et al., 2000; Wilcox-O'Hearn and Warner, 2008; Sathiamoorthy et al., 2013). In centralized storage solutions, as in RAID, the system acts in more or less the same way. The core of the RAID technology is to split (stripe) files in blocks, compute parities, and distribute the resulting blocks across disks in an array to increase reliability and performance. The term *stripe* describes a complete (connected) set of data and parity elements that are dependently related by parity computation relations. In common parlance to the coding community such collection corresponds to a *codeword* of length n. Then, the size of the stripe n may correspond directly to the number of disks in an array.

2.3.1 The "Ideal Code"

There is a certain agreement in the research community about the list of desirable properties for storage codes (Oggier, 2013). We briefly summarise here the attributes that are most mentioned in the surveyed articles for this chapter. Though, it is rare (if not impossible) to find previous work that discusses the complete features list.

High fault-tolerance. Since the main purpose of encoding data is to increase reliability, we give to this feature more consideration. Erasure codes that provide regular fault tolerance tolerate all erasure patterns that are smaller than the Hamming distance d. In his book, MacKay warned about the obsession with Hamming distance. The problem is that many code purists will focus on the codes that can guarantee to correct a well defined number of failures t, as we have discuss before, a value that is derived from d. This vision has led to define codes with bounded-distance decoders. MacKay declared that such vision is a defeatist attitude and observed that such codes cannot reach the Shannon's limit, only codes that tolerate failures beyond t can achieve it.

Codes that provide irregular fault tolerance can tolerate more failures but those are not arbitrary. The Hamming distance becomes an indicator of the minimum size of the erasures not tolerated. Since irregular codes are not MDS codes, i.e., they are not storage optimal, they are not very attractive in the theory. However, storage researchers actively investigate them as they can have interesting value in practice. In storage applications, for a given storage overhead, we look for codes that offer the highest probability that an encoded object can be recovered after a certain time.

Systematic. This subclass of encoders generate at their output exact replicas of the input. In other words, the generator matrix contains the identity matrix. This feature is certainly desirable, since it means that in the absence of failures, data can be retrieved directly. Latency and used bandwidth worsen if decoding is needed.

Low coding complexity. In a storage application, coding is performed on write requests. An efficient design has to moderate CPU overheads, disk I/O, and other resource requirements that may increase latency and/or reduce throughput.

High code rate. Code rate is the proportion of actual data (without including the encoded redundancies) transmitted by a certain code. This criterion is useful to compare codes efficiency. Yet, rateless codes (discussed in Chapter 6) do not have a fixed code rate.

As indicated by Hafner (Hafner, 2005), the term “code rate” carries connotations of digital transmissions, hence, he preferred the term “efficiency”. Although we agree with the first clause, we tend to disagree about the use of the term efficiency since the meaning is too broad. In the presence of orthogonal code techniques, efficiency is difficult to measure.

Low reconstruction cost. Read requests in a faulty environment and maintenance tasks trigger repair algorithms. It is indispensable to moderate the use of resources. The repair cost is normally computed by evaluating the network bandwidth consumed, disk I/O, the number of contacted nodes during repair, and the time to re-code a block.

Efficient degraded read. Degraded reads occur when the system is affected by failures but can still give some service. Reads in degraded mode may happen when some storage locations are not available due to scheduled maintenance. Efficient decoding for degraded read is desirable to delay recovery operations during transient failures.

Exact repair. An exact repair algorithm reconstructs exactly the missing blocks. It follows that, if the code is systematic, the property is maintained after data reconstruction.

Good locality. An (k,n-k) code has block locality r when each code block is a function of at most r other code blocks of the code. The locality of a code indicates the number of blocks needed to repair single block failures. It is therefore desirable to have small locality to reduce the repair cost.

2.3.2 RAID

RAID arrays were the first disk array organizations to use erasure coding in order to protect data against disk failures (Patterson et al., 1988; Chen et al., 1994; Schwarz and Burkhard, 1992). The acronym RAID stands for “A case for Redundant Arrays of Inexpensive Disk”. In their groundbreaking paper of 1988, Patterson, Gibson and Katz presented five settings for arrangements of independent disks to increase the performance and reliability of a system. The word “inexpensive” implies that a RAID system could be built with commodity elements. Eventually, hardware RAID controllers became highly sophisticated and, accordingly, expensive. Often, the word inexpensive is replaced by independent.

Disk arrays can increase the performance of a system because individual requests are served faster when multiple disks collaborate to increase throughput. In addition, the possibility to answer multiple requests in parallel gives the chance to increase the system's I/Os rate. Some form of redundancy is required to compensate the increase of combined disk failure rate when writing to disk arrays. RAID was proposed to increase the performance and reliability of large arrays of inexpensive disks. While RAID levels 3, 4 and 5 only tolerate single disk failures, RAID level 6 organizations use $(n-2)$-out-of-n codes to protect data against double disk failures (Burkhard and Menon, 1993). EvenOdd (Blaum et al., 1995), Row-Diagonal Parity (Corbett et al., 2004; Gang et al., 2008) and the Liberation Codes (Dongarra et al., 2009b) are three implementations of RAID level that use only XOR operations to construct their parity information. Huang and Xu proposed a coding scheme correcting triple failures (Huang and Xu, 2008). Two-dimensional RAID arrays, or 2d-Parity arrays, were investigated by Schwarz (Schwarz, 1994) and Hellerstein et al. (Hellerstein et al., 1994) who noted that these arrays tolerated all double disk failures but did not investigate how they reacted to triple or quadruple disk failures. More recently, Lee patented a two-dimensional disk array organization with prompt parity updates in one dimension and delayed parity updates in the second dimension (Lee, 2004). The codes presented in Chapter 3, simple entanglements, bear a slight resemblance to SSPiRAL (Survivable Storage using Parity in Redundant Array Layouts) (Amer et al., 2008) layouts that use simple t parity computations to provide high reliability and maintainability. Every SSPiRAL arrangement is defined by its number of unique data nodes (its degree), the number of nodes that contribute to constructing a parity node (its x-order), and the total number of nodes. For instance, a SSPiRAL layout of degree 3 and x-order 2 would use two nodes to build a parity node, and consists of three data nodes and two to four parity nodes. Researchers have explored RAID for storage at a petabyte scale (Fan et al., 2009; Fan et al., 2011).

A caveat with RAID is that rebuilds may take hours or days. Failures during rebuilds can cause permanent damage. An indicator of potential problems is the accumulation of reallocated sectors that progressively deteriorate a disk. This number can be use to build proactive defenses (Ma et al., 2015). An strategy to reduce repair time is workload distribution and parallelism. Holland and Gibson established criteria on parity declustering to maximize parallelism during recovery (Holland and Gibson, 1992), but their solution does not improve the fact that all disks are involved in the repair process. For the case of single failures, maximizing repair distribution may not be convenient unless the code has good locality.

Many agree that there is still future for collections of cheap spinning disks, however, there are practical limitations on the size of the arrays. Large arrays require redundancy and maintenance mechanisms that are expensive. It is not practical to construct large arrays of hundred or thousand disks using RAID6 unless the large arrays are further organized in small redundancy groups.

Another concern is that the market changed substantially in the last decades. The disparity between technological advances introduces more challenges. On one side, the increase on areal density causes significant drops on the price per 1 GB. Disk manufacturers continue to introduce techniques such as shingled write, heat assisted and microwave assisted magnetic recording, and bit patterned media to push for even more great storage densities (Shiroishi et al., 2009). On the other side, while disk capacity has been grown steadily, the improvements done on IOPS lay behind (Brewer et al., 2016). As a consequence, there is a bottleneck at reading and writings operations. A study on I/O-Optimal Recovery sheds some light on the trade-offs between storage overhead and I/O recovery for XOR-based erasure codes but also leaves many open problems (Khan et al., 2011).

A recent novel approach is RESAR (Schwarz et al., 2016). RESAR is focused on surviving double-failures with low storage overhead and efficient updates. It provides high flexibility for repair and its storage overhead is equivalent to a declustered version of RAID6. As stated by the authors, it has superficial similiarities to helical entanglement codes or weaver codes. Entanglement codes are designed to cope with large amounts of failures and mostly focus on permanent storage, hence an update requires re-inserting the data at the head of the chains. A point that we have in common is that we use a virtual layer to model the relationships between data and parities which is then mapped to storage devices. On the other side, the RESAR layout has minimal erasure patterns of small size like the “barbell” and the “triangle” (both of size three) that do not appear in a helical lattice.

2.3.3 Reed-Solomon Codes

Reed-Solomon codes (Reed and Solomon, 1960) are an important class of MDS codes. They are built with a dense generator matrix and since they operate over Gallois field GF(2^r), many attempts have been made to optimize the computations (Plank and Xu, 2006; Greenan et al., 2007a; Greenan et al., 2008b; Plank et al., 2013). In the context of storage, a common notation is to indicate the parameters “k” and “m” as in (k, m) RS codes.

Reed-Solomon codes became a sort of de-facto industry standard for erasure coding, and particularly to archive data. Perhaps the main reason for their broadly acceptance is because, they were already an established solution for digital communications when the storage market started to take off. Recently, they become very popular for data archival applications due to their space efficiency ratio. A limitation of RS encoded systems are the costs involved in repairing one single failure. The decoder has to obtain k fragments to recover from one erasure. This cost reflects in the number of nodes that need to be contacted, the network bandwidth overhead, and the time to repair.

2.3.4 LRC

Local reconstruction codes, e.g., (6,2,2) LRC, are implemented in Windows Azure Storage (WAS) (Huang et al., 2012). Locally repairable codes, e.g., (10,6,5) LRC are implemented in HDFS-Xorbas, a variation of the Hadoop Distributed File System at Facebook (Sathiamoorthy et al., 2013). Both are codes built on top of RS codes that support local repair by creating additional local parities. They relax the MDS constrains in order to reduce the network overhead and time to repair fragments. Besides, they slightly increase the number of erasure patterns that the code can tolerate beyond the Hamming distance. For instance, (6,2,2) LRC is capable of decoding 3 arbitrary failures. But their main focus is on the reduction of reconstruction time when only one fragment is missing.

LRC parameters may create some confusion. In spite of the identical notation, (6,2,2) LRC corresponds to a (k, l, r) LRC which is built with $k = 6$ data blocks $(x_1, \ldots, x_6)$, the data blocks are divided into $l = 2$ local groups that have one parity computed from a group of $r/l = 3$ data blocks, and $r = 2$ global parities that are calculated as in a (6,2) RS code. In the case of HDFS-Xorbas, (10,6,5) LRC corresponds to a $(k, n-k, r)$ LRC which is built with $k = 10$ data blocks, uses $n-k = 6$ parities, and has locality $r = 5$. In the next chapters, we will see that entanglement codes preserve locality while increasing considerably tolerance to non-arbitrary failures.

According to their authors, LRC are optimized for the reconstruction of data blocks in WAS but Weaver codes (Hafner, 2005), Hover codes and Stepped combination codes (Greenan et al., 2010) can reconstruct parities more efficiently. Entanglement codes are designed for optimal reconstruction of any kind of block. We found this property more realistic since failures degrade the redundancy in a storage system without conducting any selectively damage.

2.3.5 Weaver Codes

Weaver codes are (in general) non-MDS, XOR based codes designed for storage efficiency and high fault tolerance (Hafner, 2005). The code construction is described by a weave pattern that is repeated using rotational symmetry. The author presented two weaver constructions and one ad-hoc code. A featured characteristic is the constrained parity in-degree, which bounds the complexity of computations and improves the locality of the code, particularly for efficient repairs. The author strongly emphasized the benefits of the symmetrical design. Symmetry is observed at three levels:

- *parity in-degree:* every parity is computed by the same number of data;
- *data out-degree:* every data contributes to a fixed number of parities; and
- some codes have parity in-degree = data out-degree.

Weaver codes have various interesting approaches. Now, we discuss the similarities to our work. Hafner studied combinations of horizontal and vertical codes. Horizontal codes calculate parities only from data or parity blocks but never both, while vertical codes compute parities from both a group that contains data and parities. The weaver scheme permits flexible selections for the stripe size but not all values are valid. On the contrary, in entanglement codes, the stripe can have any length and it can grow by adding more storage capacity. A feature that both codes have in common is the constrained parity in-degree. Additionally, we share Hafner's approach about the symmetry in the design. But weaver codes achieve only up to 12 failure fault tolerance. The scheme to weave data and parities is not very clearly explained. The evaluation showed that only one of the constructions achieves 50% efficiency and tolerates up to 10 failures, other constructions have poor efficiency. We will show that entanglement codes can achieve higher fault tolerance in Chapter 5 and that spigot codes can increase the code rate (hence achieve higher "efficiency") in Chapter 6.

2.3.6 Flat XOR-based Codes

Stepped combination codes and HD-combination codes are flat XOR-based codes (Greenan et al., 2010). Their authors points out that flat XOR-based codes are small LDPC codes. Flat XOR-based codes are defined as erasure codes in which only one element (data or parity) is stored on a separate device and each parity is obtained from xor-ing some subset of data elements. In a follow up paper (Wylie, 2011a), Wylie described the techniques used to identify the most fault-tolerant codes in a large corpus. It considered only systematic codes with small parameter values ($k + m \leq 30$). As he explained, a brute force approach that tries all erasure patterns to decide which code is more fault-tolerant is prohibitively expensive as there are $2^{k\cdot(k+m)}$ codes and each one has 2^{k+m} possible failure patterns. Wylie devised an algorithm to determine fault-tolerance while reducing the corpus to a feasible size by noticing that there are only 2^{k+m} systematic codes and because many of them are isomorphic to one another the search space is even more reduced. In fact, his approach reduced an initial search space of 2^{81} codes to a corpus of 49,215 erasure codes. The corpus (Wylie, 2011b) includes the only MDS codes that are flat XOR-based codes (replication and RAID4) and all other non-MDS codes that are most fault-tolerant. We observe that the corpus contains only one code that has Hamming distance $d > 10$: replication with $k = 1$, $m = 10$, and $d = 11$.

2.3.7 Regeneration Codes

The cutting edge paper on regeneration codes was the first study on network coding that undertook the repair bandwidth problem (Dimakis et al., 2010). The paper presented evidence of the significant bandwidth savings of regeneration codes and a theoretical characterization of the trade-off between storage and repair bandwidth. The repair bandwidth is reduced by downloading portions of data stored in surviving nodes. This paper drove the attention of the research community and further studies have been conducted, for example (Fragouli et al., 2008; Oggier and Datta, 2011). In particular, we highlight studies that discuss regeneration codes with the interesting property of exact repair (Suh and Ramchandran, 2009; Rashmi et al., 2011). Finding coding coefficients is an active research problem. Authors from the systems community indicated that regeneration codes are not widely understood by the community; moreover, they require a complex parameterization (Jiekak et al., 2013).

2.3.8 Pyramid Codes

Pyramid codes combine local and global parities to increase read efficiency during failures (Huang et al., 2007; Huang et al., 2013). Huang observed that many codes are available with limited configurations, but pyramid codes can be constructed with arbitrary storage overhead. A problem with pyramid codes are the algebraic equations used to find code constructions. An improvement was seen in the LRC codes for HDFS-Xorbas, a subsequent work from the first author. As indicated in that work, the coding equation coefficients of pyramid codes are discovered through a search algorithm, whose complexity grows exponentially with the length of the code. Nevertheless, pyramid codes were implemented for an archival storage (Wildani et al., 2009). In this implementation, the use of global parity, denoted as "über-parity", increased fault-tolerance up to 7 more disk failures.

2.3.9 Additional Related Work

HoVer codes are XOR-based codes that combine horizontal and vertical codes and tolerate up to four disk failures (Hafner, 2006). The author presented the constructions for two fault tolerance, but showed that some parameter combinations cannot tolerate three disk failures and superficially studied the case of four disk failures. His evaluations were conducted in disk arrays in the order of 300 disks. Greenan et al. introduced large stripes using parity codes across two-way mirrored groups for non-volatile RAM (NVRAM) applications (Greenan et al., 2007b). Staircase codes combine notions of recursive convolutional coding and linear block codes and they are designed for reliable communications of streaming sources (Smith et al., 2012).

2.4 A Tale of the Storage Sector

In 1997, Douglas Engelbart received the A. M. Turing award, for more details see ACM webpage (Bardini, 2017). Engelbart was an engineer and inventor pioneer who contributed to many hardware and software innovations. A noteworthy example of his visionary ideas is his *scale change* principle. As stated at the ACM webpage, Engelbart's "scale change principle asserts that as a complex system increases in scale, it changes not only in size but also in its qualities". The scalability problem bring us directly to the challenges faced in the storage market. In fact, the storage scale changed abruptly in the last 20 years, and such change suggest us to rethink the standard solutions. The internet of things (IoT), the cloud storage industry, and high-performance computing (HPC) sector generate large volumes of data known as "big data". Large scientific research projects in the areas of genetics (Goldman et al., 2013), physics (Peters and Janyst, 2011) and astronomy (Vermij et al., 2014) need exascale storage. Failures in production parallel systems were not that common to justify investments in fault-tolerance in mid-2000s, however, in 2009 researchers recognized the urgent need of disruptive fault-tolerance mechanisms for peta/exascale computing (Dongarra et al., 2009a). Five year laters, the same authors updated their survey study indicating positive progress.

They identified five active research areas: hardware failure characterization, fault-handling standard model, fault handling improvement at all stages, programming abstractions for resilience, and standardized evaluation (Cappello et al., 2014).

Throughout these years, the research and open-source communities attempted to design scalable reliable storage systems. The following list, which is representative but by no means comprehensive, is intended to illustrate the variety of options that exist in the literature.

Cedar	Cedar is file system developed at Xerox's lab and reimplemented with a log-structure to improve robustness and performance (Hagmann, 1987).
Sprite	The Sprite log-structured file system is designed to have disk areas for free writing while using the log structure to reduce the recovery time after system crashes (Rosenblum and Ousterhout, 1992).
Elephant	The Elephant file system introduced the idea of protecting users from accidental file deletions by keeping the most important file versions of all users (Santry et al., 2000).
Oceanstore	Oceanstore is an architecture for global-scale persistent storage with two replication levels and access control via encryption keys (Kubiatowicz et al., 2000).
PASIS	Pasis architecture is designed for survivable storage systems that would provide high availablity, durability and integrity (Wylie et al., 2000).
CFS	The cooperative file system is a peer-to-peer read-only storage system (Dabek et al., 2001).
Venti	Venti is the first system that considered archival storage as a first-class citizen and it introduced the concept of write-once storage (Quinlan and Dorward, 2002).
Lustre	The Lustre file system is a parallel system often used in supercomputers (Braam and Zahir, 2002).
Farsite	Farsite is a federated scalable distributed system that assures reliability using random replication and encryption for privacy (Adya et al., 2002).
GFS	The Google file system is designed as a scalable distributed storage system to meet Google application workloads goals (Ghemawat et al., 2003).
Glacier	Glacier is a decentralized system to endure correlated failures but provide non-persistent storage (Haeberlen et al., 2005).
Ceph	Ceph distributed file system is designed to facilitate metadata management, hence, allowing objects allocation in heterogeneous object storage devices (OSDs), and to adapt to scientific computing workloads (Weil et al., 2006a).

Panasas	The Panasas distributed file system is designed to scale in performance, using parallel access to OSDs and for installations as large as 500 nodes. (Welch et al., 2008).
Tahoe	Tahoe, the least-authority file system is designed using cryptography to assure confidentiality and integrity, and erasure coding for tolerating failures (Wilcox-O'Hearn and Warner, 2008).
POTSHARDS	Potshards is a long-term archival storage with keyless security (Storer et al., 2009).
HDFS	The Hadoop distributed file system is designed for very large datasets with the goal of streaming data to user applications at high bandwidth (Shvachko et al., 2010).

Some systems are designed for mutable data while other assume immutability. Typically, an archival storage enforces a write-once policy, at least, to the subset of data that is considered immutable. Mutable data requires a consistency model to handle updates as in Oceanstore, CFS and GFS. But mutable data is orthogonal to a system that guarantees long-term durability, hence, this topic is outside the scope of this thesis. However, if resources are not unlimited, methods to claim storage are necessary even with the assumption of immutable data. Some common practices for space optimisations are coalescing duplicate blocks and the use of leases to bound the lifetime of objects. Leases can be used for space efficiency as seen in Glacier and also for reducing management overhead as in GFS. The Elephant file system proposed the use of file-grain user-specified retention policies. At the core of all strategies to build a reliable system resides the redundancy (encoding) method, which needs to be integrated in the storage architecture.

2.4.1 Reliable Cloud Storage

A general study about cloud environments was done by Armbrust (Armbrust et al., 2010). Cloud environments are prone to failures and attacks (Cachin et al., 2009). BlueSky (Vrable et al., 2012) ellaborated on the challenges when bridging the gap between client/server applications and the cloud and provided various metrics.

From the conception to the commissioning and operation of a dependable and secure system, non-trivial decisions that involve trade-offs are taken. Some design criteria and system customizations have strong impact on the provided service and its performance. However, a survey and benchmarking of cloud storage services (Dropbox, Amazon Cloud Drive, Microsoft SkyDrive, Google Drive, Lacie Wuala) showed that there was not a clear winner (Drago et al., 2013). From the user perspective, perhaps the most tricky aspect of cloud storage services is the network latency when accessing remote data. Despite the fact that cloud services are built using concepts of distributed storage systems, data usually resides in a datacenter that may be located on the other side of the globe.

A main question is the future role of cheap spinning disks in very large scale datacenters, which was addressed in a recent Google whitepaper (Brewer et al., 2016). Google's position became clear: hard drives arranged in large collections would remain due to their lower price. The author added that the growth rates in capacity/$ between hard drives and SSDs were relatively close. CERN indicated that disks are the main cause of server failures (77%) (Espinal et al., 2014).

Large arrays of disks increase throughput and system's I/O rate but the combined failure rate increases too. Many cloud-based centralized solutions use replication or RAID-like techniques, the latter often built with erasure coding (Plank et al., 1997). Normally, data is initially replicated, then erasure coding techniques are applied in the background. Low latency requirements are relaxed as data ages; thus, in many applications replicas are deleted. Using erasure codes to archive data brings considerable storage savings. According to the literature, the common settings for erasure codes have less than 2x redundancy.

A problem with erasure coded data is that nobody wants to spread coded data across datacenters due to the inefficient use of bandwidth during repairs. According to L. Barroso (personal communication, March 11 2016), it is preferable to have enough erasure coded blocks in one locale to be able to retrieve data efficiently. But, erasure codes cannot cope with network partitions if data is confined to one location. Local-area and wide-area networks failures commonly occur, although wide area network failures are less documented (Bailis and Kingsbury, 2014). To support datacenter failures, Google and Facebook combine hybrid redundancy schemes. For instance, Google uses multi-cell replication to survive datacenter failures, and RS encoding for local failures (Ford et al., 2010). The Facebook f4 system uses geo-replicated XOR-based codes to provide datacenter fault-tolerance, and RS encoding within a single datacenter (Muralidhar et al., 2014). RS encoding is storage efficient, although when used in combination with replication the overall storage overhead is significant.

Many storage solutions have bandwidth constraints that put limitations on the code parameter values. A system that uses *RS(k,m)* codes tolerates m failures, but requires k I/O accesses and $k * B$ bandwidth to repair a single failure of B bytes, where B is the fixed size of the block. Many experts attempted to reduce the repair overhead(Papailiopoulos and Dimakis, 2014; Khan et al., 2012; Huang et al., 2012; Sathiamoorthy et al., 2013). To avoid serious performance issues, real-world systems advocate small values: RS(6,3) in Google single-cell scheme (Ford et al., 2010), RS(10,4) in Facebook f4 system (Muralidhar et al., 2014) and $k + m \leqslant 20$ in Microsoft Azure (Huang et al., 2012). These practical limitations have consequences: the code characteristics can change radically. For example, theoretically, optimal Reed-Solomon codes can protect data against massive correlated failures, but, in practice they can tolerate only a few failures. A concern is that time between disk replacements has significant levels of correlations as it was shown by experts (Schroeder and Gibson, 2007). Power outage and other sources of temporary failures cause correlated failures in real storage systems (Chansler, 2012; Dean, 2010). Although temporary failures may not cause permanent data loss, high availability applications need redundancy methods that can cope with these failures too. Indeed, systems can keep data safe only with the existence of continuous monitoring and repair operations that increase the system cost. Unfortunately, when discussing redundancy methods, most of the arguments focus only on the storage cost.

2.4.2 Approaching Eternal Storage

Cold storage was initially associated with tape backups stored in at least two places to avoid hazards. The physical media was a finite resource and had a limited access, i.e., characterized by sequential access to data. Usually, a small collection of cartridges were used in a customized backup schedule that typically combined full and incremental backups and reused the media according to retention policies. It was an error-prone manually process and automated solutions turned to be expensive for small enterprises.

Two paradigm shifts assured the position of archival storage as a first-class citizen. One was the design of worldwide scale distributed systems based on commodity hardware. The proliferation of cheap storage devices and the bureaucracy associated with time sharing on large computers facilitated the increasing popularity of distributed systems. Plan 9 (Pike et al., 1995) is a remarkable distributed system created in the mid ’80s by Bell Labs. A second paradigm shift was to consider that storing data eternally is possible. Stable storage is associated with write-once read-many (WORM) devices. Plan 9 engineers considered a WORM jukebox as an “unlimited resource”, in other words, it was expected that a full jukebox would be replaced by a new jukebox with improved technology, hence, more storage capacity. Archival storage gained significant importance due to the vertical and horizontal expansion of storage facilities during that time. Venti, a block-level network storage system intended for archival data, soon became a model for other systems, which offered similar features or slightly variations.

The features provided by an archival storage can vary considerably and even the boundaries between data archiving and data storage become diffuse. For example, Venti specifications promise access time comparable to non-archival data. Another example is the PASIS architecture, a study done by Carnegie Mellon fellows on *survivable information storage systems* to provide durability, integrity and high availability. *Durability* clearly depends on the redundancy methods used in the system as it was previously discussed. *Integrity* is a feature that is worth to discuss in detail, we do that in the next subsection. We observed that *high availability* is not typically considered in archival systems although it becomes increasingly necessary to provide good access to data that ages with time. For example, Facebook created the f4 system (Muralidhar et al., 2014) dedicated to manage warm data. Their work showed the benefits of treating data differently according to its temperature. Data is assigned a temperature level (hot, warm or cold) according to its usage pattern. The considerable amount of data in the warm category (over 65 PBs) is a good reason to design a system optimized for this category. Another interesting approach for achieving high availability is the recent paper on the scheme “procode”, a proactive erasure code that combines disk health monitoring and adjusts the replication factor accordingly (Li et al., 2016).

Eternal storage is often discussed by librarians in the context of digital preservation systems. The field of digital preservation sets standards for ingesting and disseminating information (Rosenthal et al., 2005). Preserving content means that the format in which the data was encoded will remain valid and readable in future, otherwise the format must be updated. The LOCKSS project, "lots of copies keep stuff safe", initiated in 1999 at the Stanford University Libraries to safeguard digital assets. They developed a peer-to-peer system that replicates data across peers, has self-healing mechanisms and protectes the system against free-loading and theft (Maniatis et al., 2005). The Internet archive project is a non-profit public digital library that it crawls the Internet to backup website pages among other services. Researchers documented that its architecture was maintained by only five employees at least until 2009 (Jaffe and Kirkpatrick, 2009). A big concern in the community is how to keep the cost low and how to estimate the cost of long term storage (Rosenthal et al., 2012). Another issue is the risk of censorship attacks. Many proposals from the academic and open-source community focus on anti-censorship measurements (some of them will be discussed in the next section).

Numerous studies have been conducted in unconventional storage media such synthesized DNA or crystals. We do not discuss them since none of them seem to be economically feasible in the short term.

2.4.3 More Challenges and Open Problems

Building a reliable storage system is a complex task. As we strive for designing a code that is practical and flexible we must insist that storage savings are only one angle of all the existing challenges and potential issues that may appeared in a long-term storage system. These are significant problems that are often overlooked.

Data Scrubbing

Data scrubbing is a process to check for any kind of errors and inconsistencies and to correct them before data loss occur. Scrubbing has a positive impact on solving failed disk sectors and reducing the probability of silent data corruption. Just reading data helps to verify its existence. In addition, checking data signatures helps to detect corruption. All data must be scrubbed frequently to maintain a consistent acceptable level of reliability. But the process consumes energy, may reduce performance and scrubbing one disk may take hours.

Researchers studied how frequent should it be. An opportunistic policy, which only scrubs powered-on disks three times a year on average, was found to be the most efficient for archival systems (Schwarz et al., 2004). In archival systems, hard disks are most of the time powered off. The scrubbing is activated if the disk is powered on to access some data. The process may deteriorate read requests performance but low latency expectations in archival systems are relaxed. On the contrary, general purpose storage systems usually have their disks powered on all the time. In that case, scrubbing plans avoid to impact system's performance, i.e., the process is activated only during disk idle time. According to T. Schwarz (personal communication, April 10 2017), one of the assumptions for that study was that additional disk utilization would cause more errors but that is probably not an important factor. However, he adds, disks with constant high utilization tend to have a higher degree of failure. Some researchers suggested that one scrubbing per week was not enough to reduce the errors adequately and presented some solutions to reduce the rate of undetected data by two or three orders of magnitude (Rozier et al., 2009). Scrubbing models depend on statistical data which is hard to obtained. A study from Google presents failure characteristics in large disk populations and the impact of self-monitoring analysis and reporting technology (SMART) parameters (Pinheiro et al., 2007). The authors conclude that an accurate predictive failure model should use signals that go beyond the SMART parameters.

The amount of data that can be corrupted is becoming more significant with the increasing on disk area capacity. A study conducted by CERN engineers showed that, in normal readings, RAID controllers do not check data integrity by reading the parity (Panzer-Steindel, 2007). The disks became 3x more stressed when the command to verify data was executed.

Why data scrubbing matters? The storage sector is interested in understanding workload (Riska and Riedel, 2006; Basak et al., 2016). In particular, disk manufacturers target disks to different segments according to their *annualized workload rate*, for example, a Seagate disk enterprise capacity (nearline) is designed for heavy workload rate with limit at 550 TB/year, while disks for light workload rate have a 180 TB/year limit. T. Feldman (personal communication, May 4 2016) mentioned that sometimes clients that used light workload rate applications to write cold data chose instead heavy workload rate only due to their scrubbing policies. It remains an open question if high fault tolerance codes could effectively reduce the frequency of scrubbing. A positive answer may lead to promising savings in drive acquisition as well as lower bandwidth capability, and further reduce the operating costs by reducing energy consumption caused by excessive scrubbing work.

Integrity

To avoid misunderstandings, we must emphasize that there are two main types of data corruption in the context of integrity. One type of data corruption is the silent data corruption that disk scrubbing tries to detect. A second type of data corruption is due to malicious behaviour. WORM devices help to protect data when an attacker has limited physical access. But more powerful attacks might be conducted by a disgruntled employee or a dictatorial government. Generally speaking, an attacker needs to modify data, metadata and all the redundant data stored in the system to make data tampering undetected. We can assert, without any specific attack model, that tampering is relatively easy if data is less redundant and the storage architecture is more centralized. Since the majority of storage systems only keep one or two copies of data tampering in many cases is straightforward, i.e., tamper could occur without being noticed. Log analysis creates high overhead and is not always useful.

Tamper-proof storage usually assumes that a fingerprint of the content, i.e., a distinctive identifying characteristic, is known so that integrity can be verified. However, this assumption is not straightforward to guarantee under certain environments. Yet, without fingerprints, how does a third-party verify that certain content was not tampered either by the data owner or by the data holder?

In the next section, we discuss the original work on entanglement as a method to make tampering with data difficult. One important question to address for storage systems is if the system allows undetectable modifications. This kind of problem is commonly addressed using cryptographic solutions. Building a tamper-evident mechanism using emergent properties of the coding technique is a novel approach. The concept of entangled data is promising since it makes more difficult to modify data undetectably. It opens the opportunity to build a tamper-evident mechanism.

Economics

We have already discussed various aspects of the storage economics: the role of cheap spinning disks in datacenters, market segmentation according workload and the pervasive obsession with the storage efficiency in the design of redundancy methods. By pervasive obsession, we mean that there are other costs that remain hidden when we only focus on capacity savings. For instance, maintenance cost may constitute a hindrance to archival projects. A storage system that guarantees high reliability with a low replication factor, i.e., low fault-tolerance, must have important maintenance cost. It is fair to assume that if the system had more fault-tolerance it would delay repairs and reduce maintenance operations. Overall, there is little information on this subject. To give a rough idea, Microsoft published that the cost estimation of hardware repairs (mostly due to hard disks failures) was over a million dollar for a datacenter with more than 100,000 servers (Vishwanath and Nagappan, 2010). Some researchers studied replication strategies for global-scale distributed system to design a low cost replication maintenance strategy. They proposed Carbonite, a wide area replication method that maintains replication with low network traffic (Chun et al., 2006). A concern is that their assumptions seem unrealistic as they considered uncorrelated failures only.

Another important question to ask is whether the cloud is economically feasible for the perspective of a client interested in long-term storage. Rosenthal's conclusion is negative, the price of cloud services is expensive for the long-term (Rosenthal et al., 2012). He showed that cloud services did not reduce their prices fairly, according to the historical decreasing trend in raw disk prices. Clients easily become caught between the storage price and the bandwidth price since the cost of data migration to a different provider could become prohibitive for an archival storage project. It has became a world wide concern that although cloud storage systems take advantage of decentralized algorithms, in a sense, the cloud is too centralized.

Decentralized cooperative storage models such as the LOCKSS project are more interesting solutions for long-term storage. In cooperative storage, each node shares some resources, i.e., bandwidth, storage and or computation. Peer-to-peer based storage solutions have been widely experimented by the industry, researchers and the open source community, some examples are: Oceanstore (Kubiatowicz et al., 2000), CFS (Wylie et al., 2000), Farsite (Adya et al., 2002), Freenet (Clarke et al., 2001), Gnutella (Ripeanu, 2001), Wuala (Mager et al., 2012), and more recently IPFS (Protocol Labs, 2014). But, in general the peer-to-peer model has gradually declined in favor of centralized solutions. Perhaps two reasons that caused its decline were the lack of incentives to node's owners for sharing resources and the lack of an effective redundancy method for distributed networks. The first problem could be solved with novel blockchain and smart contract technologies (Christidis and Devetsikiotis, 2016; Kosba et al., 2016). The second issue underlines the relevance of research on new encoding methods.

Redundancy methods lie at the root of achieving high availability using unreliable nodes. Replication is widely used due to the high bandwidth requirements of erasure coding, especially to rebuild single-failures. In addition, it is easier to use replicas as a strategy to improve the download time of popular files. The simplest approach is that any node that participates in the path to download a file may keep a copy of the file to become a new source. On the contrary, unpopular files have less redundancy and they are likely to disappear as the file ages. Such behavor is a “natural” strategy to claim space. For long-term storage, this strategy does not make sense since old files tend to be unpopular.

Better strategies to replicate files in peer-to-peer networks are needed. Researchers identified trade-offs between high reliability and cross-system bandwidth (Blake and Rodrigues, 2003). A problem is the bandwidth cost of node departures, which cause data transfer to other nodes. Another problem is the sheer amount of replicas that are necessary for high availability. The authors found that to provide high availability using unreliable nodes was 120 replicas, even for erasure coding the redundancy was high (equivalent to 15 replicas). We noticed that centralized architectures suffer the same trade-offs but a big difference is that centralized solutions often limit redundancy to three times the storage overhead or even lower values for erasure coded data.

Block Placements

Storage systems use mapping algorithms to store and locate blocks according a placement policy and the available resources. The search for a scalable solution has been studied in three main directions: table-based, rule-based and pseudorandom-based distribution. Advantages and disadvantages of each strategy are indicated in (Miranda et al., 2014). A common solution is a pseudo-random placement based on a hash algorithm but some hashing methods produce unbalanced environments and waste resources. The “balls into bins” model is a simple approach to study the efficiency of a placement scheme. Suppose that the system has m balls and n bins. If the task is to place each ball into a bin by choosing the destination at random, the maximum number of balls in any bin, can be estimated using the formulas presented in (Raab and Steger, 1998).

Why placements matter to the design of practical codes? Because distributions impact on system's reliability. A caveat is that very little is known about how placement policies affect the fault tolerance of different encoding methods and more studies are required.

More Technicalities

The decisions done during the design and implementation of a storage system involve compromises. Finding good compromises is a difficult task, especially for systems that behave dynamically. For example, one critical decision is to choose the right code parameters for RAID-like systems. Code parameters (k, m) RS are not variable: the system must re-code all data if k and m change. Thus, changing reliability requirements may become extremely expensive. Reed Solomon codes, as well as other codes based on them, are not flexible in their parameters.

Another problem is the space overhead created by small files encoded with systems using large k values. This well know problem occurs because many solutions apply the same redundancy settings to all documents to simplify data management. In summary, encoding methods that have flexible parameters may simplify decisions during system's design and implementation and improve the system's scalability.

Sustainability

The main difference of the cloud industry from the HPC sector is that the cloud providers want to provide a reliable service using cheap unreliable components instead of expensive supercomputers. This method is also attractive to the HPC sector as it allows substantial economies of scale. In fact, a hot research topic is the HPC and Big Data convergence discussed by Carretero in a recent lecture (Carretero, 2017). He highlighted that sustainability is a major challenge in large-scale complex systems. He further add that sustainability should be the result of a "holistic approach to manage the whole ecosystem"and that sustainability metrics are needed to understand how all the factors (data management, programmability, resilience, energy efficiency, scalability, etc.) affect the system.

2.5 Data Entanglement

Censorship resistance and storage systems that provide a reliable service in the presence of unreliable nodes are topics widely investigated, for example: the eternity service (Anderson et al., 1996), Publius (Waldman et al., 2000), Freenet (Clarke et al., 2001), the free haven project (Dingledine et al., 2001), and Farsite (Adya et al., 2002). Data entanglement was a novel approach that proposed to break the one-to-one relationship between a block and a document and turn it into a one-to-many relationship, i.e., one block becomes part of many documents. Tangler (Waldman and Mazieres, 2001) and Dagster (Stubblefield and Wallach, 2001) used entanglement as an anti-censorship mechanism in document storage systems. As suggested by the researchers who undertook the first theoretical framework for data entanglement (Aspnes et al., 2007) an efficient entanglement method is not straightforward to devise.

We observe that entanglement introduces extra redundancy to the system and, hence, also contributes to its robutness. We investigated different novel approaches to entangle data efficiently with the objective to achieve high levels of document dependency and use the created redundancy to increase file durabillity. The first outcome was *helical entanglement codes*, an entanglement method that creates interdependencies between content to protect data against failures (Estrada-Galinanes and Felber, 2013). Since then, we have been actively researching data entanglement to propagate redundant information across a storage system and provide multiple paths to recover unavailable content. The results of this study are documented in this thesis.

2.5.1 Tangler

Tangler (Waldman and Mazieres, 2001) consists of three main components: a publisher program that breaks the file in fixed-size (16K) blocks, a reconstruction program that reconstructs the file from fetched blocks, and a server daemon that distributes and retrieves blocks. The system assumes that all participating servers know each other and interact in a peer-to-peer fashion. Tangler uses consistent hashing to route queries among servers and relies on cryptographic hashes (SHA-1 and hash trees) to certify data published by the servers. Participants can inject blocks in a public pool that feeds the entanglement algorithm. The publisher program performs the entanglement using polynomial interpolation. The entanglement input is two random blocks from the public pool and one data block of the file to produce two new server blocks that are injected to the pool. A data block is reconstructed with any three of the four associated server blocks.

2.5.2 Dagster

Dagster(Stubblefield and Wallach, 2001) has three architectural components: an anonymous channel between clients and servers, an out-of-band channel for document announcements, and the publishing server. Dagster's XOR-based entanglement model proposes to intertwine files with a number of pre-existing documents (c factor) while constructing a direct acyclic graph of dependencies. The user can announce the document after some time with the hope that legitimate blocks are linked with the file's blocks. When retrieving a document, the user has the additional cost of reading all the c documents that were used in the entanglement process.

2.5.3 Theoretical Framework

A study on document entanglement (Aspnes et al., 2007) observes the limitations of Tangler and Dagster, in particular their weak share-block notion of entanglement. Deleting a document affects only a small number of entangled files in Dagster and almost none in Tangler. The study introduces a theoretical framework for entanglement and defines two interesting concepts:

1. *document dependency*: if document d_i depends on document d_j, then whenever d_j cannot be recovered, neither can d_i;
2. *all-or-nothing integrity*: a storage system is all-or-nothing when document dependency exists between every d_i and d_j, i.e., the system provides maximum achievable dependency.

2.5.4 Network Coding

Mixing data is also used for network coding. In network coding, intermediate nodes are allowed to send linear combinations of packets previously received to increase robustness and potentially increase network throughput (Fragouli et al., 2006). One immediate application of network coding are sensor networks. Since the initial proposal of network coding as a solution for distributed storage (Dimakis et al., 2006), several studies have been conducted, for example (Gkantsidis and Rodriguez, 2005; Kamra et al., 2006; Zhang et al., 2006; Wang et al., 2006). Dimakis and other co-authors conducted a review of the literature on this topic (Dimakis et al., 2011).

2.5.5 Entangled Cloud Storage

Our initial work on entanglement codes (Estrada Galinanes, 2013) is to the best of our knowledge the first work that proposed data entanglement for permanent storage data in a cloud environment prone to failures and attacks. A complete section of our full paper on helical entanglement codes (Estrada-Galinanes and Felber, 2013) describes how entanglement codes may be used in a distributed storage system. It presentes the architectural components of such a system and the interactions between the different parties involved in the storage and retrieval operations. More details on helical entanglement codes are given in Chapter 4. Our recent ideas towards entangled storage systems are discussed in Chapter 7.

The security aspects of cloud storage attracted the attention of the cryptographer community. The paper titled "Entangled Cloud Storage"(Ateniese et al., 2016) presents an entangled cloud storage based on the notion of all-or-nothing integrity. The entangled encoding output has the size of the encoded data when the file is large enough, though it is not clear what happens with small files. Their discussion about practical implementation includes content previously discussed in our architecture such integrity and coordination. Overall, their solution is far away from being practical and there is no evidence that it can be implemented in a real cloud storage.

A contemporary research approach is step-archives (Mercier et al., 2015). Step-archives combines data entanglement and erasure correcting codes for archiving data permanently with the goals of data integrity and tamper resistance. The authors studied the resilience to attacks of two models: proximity entanglement and random entanglement and provided a comparison between both approaches. Proximity entanglement uses a sliding window that limits which set of documents are used during the entanglement process. The sliding window seems identical to the notion of "lead window", presented in our work for helical entanglement codes and briefly discussed in Section 7.3.4.

2.5.6 Friendstore

Friendstore (Tran et al., 2008) is a cooperative backup system based on trusted nodes. It proposes mixing user's data using a redundancy scheme called xor(1,2). It provides simultaneously redundancy for documents owned by different users. The novel idea in this paper is to trade-off bandwidth for storage by mixing data. Consequently, it only encodes data when disk space is limited.

2.6 Summary

For several decades, reliable digital communications received much attention from information theoreticians. Meanwhile, coding techniques received considerable interest by system's engineers that build reliable storage systems. Despite the shared interest on reliability and coding, there is a huge gap between the information theory, storage and cryptography communities. The problems we are solving have many angles and belong to an interdisciplinary domain but when treated independently it becomes more difficult to design solutions under realistic assumptions.

In more detail, the implementation of efficient redundancy mechanisms in complex storage systems is challenged by the existence of problems not considered in the theory. In addition, the scale change principle asserts that as a complex system increases in scale, it changes not only in size but also in its qualities. A growing body of literature has examined the storage overhead and repair costs of redundancy schemes but neglected other challenges for storage systems at the exabyte scale. Very little is known about the impact of data scrubbing in the reliability of storage systems or about the hidden cost of current redundancy mechanisms, e.g., hybrid redundancy schemes to tolerate network partitions or the maintenance cost to assure a high reliability in spite of implementations that have low fault-tolerance.

The code construction approach based on algebraic coding became less popular in the information theory field but it is often used to find codes for storage applications. The asymptotic properties of LDPC codes do not work well in many storage applications since they use short length codes (small k). Finding good code parameters is still a black art. Computer scientists tried exhaustive computation approach to find codes with high fault-tolerance. Ad-hoc code constructions are abundant in literature although their impact is arguable due to its lack of flexibility.

To conclude, we could not find in the literature codes that offer the combination of high fault-tolerance and flexibility that entanglement codes provide. Mixing data is a promising solution to protect data against failures and attacks and increase performance in permanent storage systems.

3 Simple Entanglements

This chapter presents the study of open and close entanglements, two simple data distribution layouts for log structured append-only storage systems. Both techniques use equal numbers of data and parity drives and generate their parity data by computing the exclusive-or (XOR) of the most recently appended data with the contents of their last parity drive. While open entanglements maintain an open chain of data and parity drives, closed entanglements include the exclusive-or of the contents of their first and last data drives. We evaluate five year reliabilities of open and closed entanglements, for two different array sizes and drive failure rates. Our results show that open entanglements provide much better five-year reliabilities than mirroring, and reduce the probability of data loss by at least 90 percent over a period of five years. Closed entanglements perform even better and reduce the probability of data loss by at least 98 percent. This chapter is largely based on our publication in IPCCC'16 proceedings (Estrada-Galinanes et al., 2016).

3.1 Introduction

Despite recent advances in solid state storage, the lower cost of magnetic disk drives ensures that they remain today the most widespread storage medium in large data centers. One of the main disadvantage of these drives is their poor reliability (Bairavasundaram et al., 2007; Beach, 2014; Pinheiro et al., 2007; Schroeder and Gibson, 2007). As a result, all disk-based long-term storage solutions incorporate enough data redundancy to be able to reconstruct the data stored on any individual drive. These solutions are as diverse as mirroring, RAID level 5 (Chen et al., 1994; Patterson et al., 1988), RAID level 6 (Schwarz and Burkhard, 1992; Burkhard and Menon, 1993), two-dimensional RAID arrays (Hellerstein et al., 1994; Schwarz et al., 2016), and SSPiRAL arrays (Amer et al., 2008).

Entanglements trade space for increased reliability and faster updates, especially in the case of log-structured append-only storage systems. Simple entanglements require equal numbers of data and parity drives; therefore, they have the same space overhead as mirroring. In counterpart, a simple open entanglement chain with $2n$ drives will tolerate the failure of any of its drives and the simultaneous failure of any two of them, except for the two last drives, which is much better than a mirrored organization. At the same time, appending a block to the entanglement will require one read and two writes while RAID level 6 and two-dimensional RAID arrays will require two reads and three writes.

We present here a full probabilistic analysis of the reliability offered by simple entanglements. We note first that entanglement chains can be closed to increase their reliability and show that the conversion process is both fast and reversible. We then model each entanglement as a Markov chain under standard stochastic assumptions and use our model to investigate the behavior of open and closed entanglement chains for four different scenarios, namely:

1. A small array consisting of 20 fairly reliable drives with an mean time to failure (MTTF) of 200,000 hours. These disks would fail at the rate of 4.28 percent per year.
2. The same array with drives having an MTTF of only 35,000 hours. These disks would fail at the rate of 25 percent per year.
3. A medium-size array consisting of 50 fairly reliable drives with an MTTF of 200,000 hours.
4. The same array with drives having an MTTF of only 35,000 hours.

Our results show that open entanglements provide much better five-year reliabilities than mirroring and reduce the probability of a data loss by at least 90 percent over a period of five years, which corresponds to the maximum useful lifetime of most consumer-class drives (Beach, 2014). Closed entanglements perform even better and reduce that probability by at least 98 percent.

The remainder of this chapter is organized as follows. Section 3.2 introduces simple entanglements. Section 3.3 discusses possible array organizations. Section 3.4 discusses the vulnerability of open and close entanglements to double, triple, and quadruple drive failures. Section 3.5 introduces our Markov model and presents the results of our investigation. Section 3.6 sketches possible extensions, Section 3.7 presents a summary.

3.2 Simple Entanglement Layouts

The idea behind simple data entanglement is to intertwine data and parity blocks using the bitwise XOR operation with the goal of *refreshing* previously calculated parities and increasing the scope of redundant information. The process builds a chain that associates old information to the new data. In this way, old blocks will become indirectly dependent on blocks that will be inserted in future without the need to run the algorithm more than once.

A pseudocode description of the simple entanglement encoder algorithm is shown in Algorithm 3.1. The encoder has time complexity $\mathcal{O}(n)$ with n being the number of data blocks that

Algorithm 3.1 — Simple entanglement encoder.

```
1: $pool$: Queue containing blocks ready to be encoded

2: while $\neg$IsEmpty($pool$) do
3:     $u \leftarrow$ GetParity()                ▷ Get from cache
4:     $v \leftarrow$ Dequeue($pool$)            ▷ Get oldest element
5:     $w \leftarrow$ Entangle($u, v$)           ▷ Compute $u \oplus v$
6:     WriteData($v$)                            ▷ Write to array
7:     WriteParity($w$)                          ▷ Write to cache and array
8: end while
```

are entangled. The space complexity is $\mathcal{O}(2n)$, which is equivalent to say that the storage capacity is halved. In other words, simple entanglements cause an storage overhead equivalent to mirroring. To improve performance, the encoder uses a write-through cache to keep in memory the last parity used in the entanglement. Data blocks are kept in a queue and they are encoded sequentially per their arrival order. We use the following notations:

d_i	is a data block. Any object that is stored in the system is split in one or more d_i. The index i indicates the order in which the blocks are entangled and their position in the chain.
$p_{i,j}$	is the parity block associated with data blocks d_i and d_j.
D_i	is a drive that stores data blocks.
P_i	is a drive that stores parity blocks.

The general expression that defines a data entanglement is

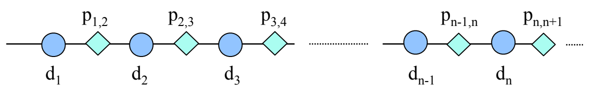
$$p_{i,i+1} \leftarrow d_i \oplus p_{i-1,i}, \tag{3.1}$$

for i greater than zero.

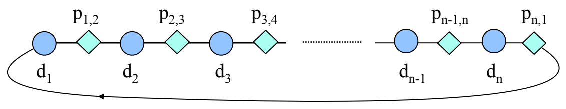
We define two classes of entanglements, namely, open and closed entanglement chains (Figure 3.1).

3.2.1 Open Entanglements

The encoder algorithm creates a never-ending chain, see Figure 3.1a. As long as space is available, the encoder will keep adding entangled blocks at the right extremity of the continuously growing chain. When the encoder starts for the first time, there is no parity block $p_{0,1}$ to use on the right part of Equation 3.1. The encoder will instead copy d_1 into $p_{1,2}$, because it is equivalent to assuming the existence of a fictitious parity block $p_{0,1}$ that only contains zeroes. The value of the next parity block, block $p_{2,3}$, is computed using Equation 3.1, that is $p_{2,3} \leftarrow d_2 \oplus p_{1,2}$.



(a) Open Entanglements



(b) Closed Entanglements

Figure 3.1 – Open and closed entanglement chains

Similarly, the decoding equations are derived from Equation 3.1. They result from the associative property of the exclusive-or (XOR) operator and the iterative process that entangles blocks while building the entanglement chain. If a data block is not available, it can be rebuilt using Equation 3.2,

$$d_i \leftarrow p_{i-1,i} \oplus p_{i,i+1} \, . \tag{3.2}$$

In addition, there are two ways of recovering parity blocks. One way is using Equation 3.1 and the other is using Equation 3.3:

$$p_{i,i+1} \leftarrow d_{i+1} \oplus p_{i+1,i+2} \, . \tag{3.3}$$

Note that this second expression does not apply to the last parity block $p_{n,n+1}$ of an entanglement chain as neither d_{i+1} nor $p_{i+1,i+2}$ exist. As a result, the content of the last data block d_n will be irrecoverably lost if both blocks d_n and $p_{n,n+1}$ are lost.

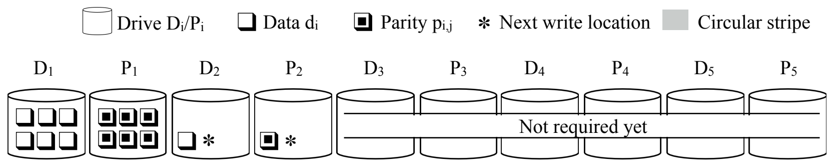
3.2.2 Closed Entanglements

Closing the entanglement chain eliminates this fatal double failure by entangling the last data block d_n with the first data block d_1. A closed entanglement is built exactly as an open entanglement until there are no more data to be added and the chain can be closed, see Figure 3.1b. When this happens, the content of the first parity block is recomputed using Equation 3.4:

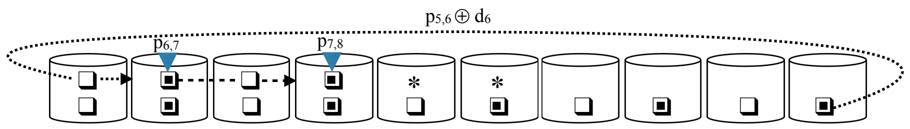
$$p_{1,2} \leftarrow d_1 \oplus p_{n,1} \, , \tag{3.4}$$

where $p_{n,1}$ is the last parity block of the entanglement. As a result, the content of $p_{n,1}$ can now be reconstituted using $p_{n,1} \leftarrow d_1 \oplus p_{1,2}$.

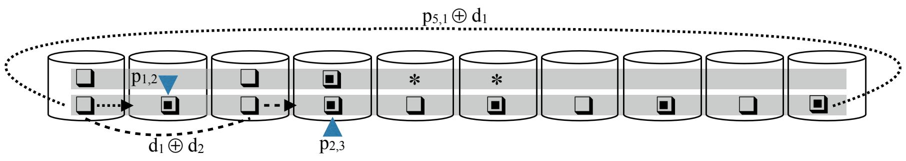
The content of parity $p_{2,3}$ will remain unchanged but it will now be defined as $p_{2,3} \leftarrow d_2 \oplus d_1$,



(a) Full-partition write with unknown entanglement class (not yet defined)



(b) Block-level striping with open entanglement chain



(c) Block-level striping with closed entanglement chain

Figure 3.2 – Array organizations in combination with open and closed entanglements

where d_1 replaces $p_{1,2}$.

As the conversion process is $\mathcal{O}(1)$ with respect to the length of the chain, closing an open entanglement is both fast and easy. The reverse process is even simpler: it only requires overwriting the content of parity block $p_{1,2}$ with the content of data block d_1. As a result, it is always possible to add elements to a closed entanglement chain by reopening it.

3.3 Array Organizations

The entanglement algorithm presented in the previous section organizes blocks into a log. There are several choices for writing the blocks to disks. We sketch here two possible array organizations: full-partition write and block-level striping. As both organizations will require equal numbers of data drives and parity drives, all arrays will have an even number of drives.

3.3.1 Full Partition Write

In this approach, blocks are written sequentially on the same drive. The process does not spread content across drives. Instead, it waits until the current drive is full before writing into a new drive. This data layout is best suited for archival applications with very low read to write ratio. Most of the drives will remain idle and can be powered off as in a massive array of idle disks (MAID) configuration (Colarelli and Grunwald, 2002), which could result in significant energy savings.

Another advantage of the approach is its scalability as drives can be added to the array at any time. The minimum requirement is two drives, but this would correspond to a mirror configuration. To create a true entanglement chain, at least four drives are needed. In addition, a closed entanglement chain that can tolerate all double drive failures would require at least three data drives, for a total of six drives.

The main limitation of this data layout is its write penalty as three drives are involved in the operation. To complete a write, the system needs to read a parity from an extant parity drive, XOR that parity with the new data and write the new data block and the new parity block on two different drives. A new parity is computed for every write. The general encoder, Algorithm 3.1, uses a write-through cache to avoid the extra read. The problem is that we would need a cache large enough to store the contents of a whole parity drive in order to eliminate the read penalty.

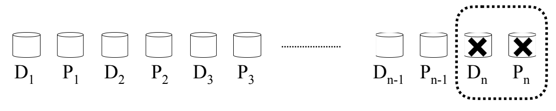
Figure 3.2a shows an example of a ten-disk array with four active drives. As writes only involve the last two most recently written drives of the array, the array could grow indefinitely if the application requires it.

3.3.2 Block-level Striping

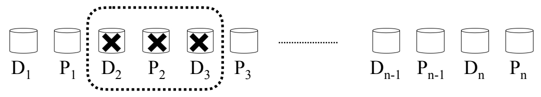
This second approach distributes data over all available drives to improve performance. New content is split into data blocks of size b. These data blocks are then spread across some or all of the n data drives. Block-level striping requires to decide between an open and a closed entanglement when the array is set up. Figure 3.2 shows instances of both organizations. Their main difference is the way parities $p_{1,2}$ and $p_{2,3}$ are calculated. In an open chain, unless otherwise stated, the elements stored in the same disk or partition are part of the same growing chain. Elements located at n hops are written to the same disk. Figure 3.2b shows how to calculate $p_{6,7}$. It is the second element stored in drive P_1. The figure also shows how to use $p_{6,7}$ to compute $p_{7,8}$, which is stored in P_2. Special elements $p_{1,2}$ and $p_{2,3}$ are calculated only once. Figure 3.2c shows how to calculate the special elements in a closed chain. Each closed chain forms a circular stripe. One of the chains is not yet finished, therefore, its element $p_{1,2}$ is not yet calculated. As a result, all blocks stored in the same disk will be part of independent stripes.

3.4 Vulnerability Analysis

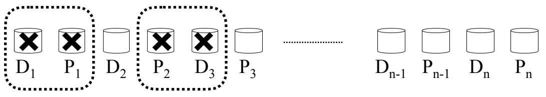
In this section, we will evaluate the probabilities that open entanglements, closed entanglements and mirrored organizations will not be able to tolerate double, triple and quadruple drive failures. Our focus will be on entanglements using the full partition write approach as



(a) Open entanglements - Type A failure



(b) Open entanglements - Type B failure



(c) Open entanglements - Type C failure

Figure 3.3 – The three irreducible fatal failure patterns of open entanglements

it simplifies our analysis. In all three cases, we will begin by searching for irreducible failure patterns, that is, the specific failure patterns that involve a minimum number of failed drives (Corderí et al., 2010).

3.4.1 Open Entanglements

As we can see on Figure 3.3, open entanglements exhibit three irreducible failure patterns, namely:

1. The failure of the last data drive of the entanglement, say, drive D_n in our example, and its associated parity drive P_n. We will call this failure a *type A* failure.
2. The failure of two consecutively numbered data drives, say, drive D_2 and D_3 in our example, and the parity drive in between the two failed data drives, that is, parity drive P_2. We will call this failure a *type B* failure.
3. The failure of two data drives, say data drives D_1 and D_3 in our example, and all the parity drives between them, say parity drives P_1 and P_2 in our example. We will call this failure a *type C* failure.

In all three cases, the array will lack the information to reconstruct the content of the failed data drive(s). Consider now an open entanglement with $2n$ drives and assume it experiences the simultaneous failure of two of its drives. Out of the $\binom{2n}{2}$ possible double failures, only the type A failure will result in a data loss. As a result, the probability α that the entanglement will not tolerate a double drive failure is:

$$\alpha = \frac{1}{\binom{2n}{2}}\,.$$

The triple failures that will result in a data loss include:

1. The type A double failure mentioned above combined with the failure of any of the $2n-2$ remaining drives.
2. Any of the $n-1$ type B failures mentioned above.

Hence, the probability β that the entanglement will not tolerate a triple drive failure is:

$$\beta = \frac{3n-3}{\binom{2n}{3}}\,.$$

The quadruple failures that will result in a data loss include:

1. The type A double failure mentioned above combined with the failure of two of the $2n-2$ remaining drives for a total of $\binom{2n-2}{2}$ fatal quadruple failures.
2. Any of the $n-1$ type B failures mentioned above combined with the failure of any of the remaining $2n-3$ drives for a total of $(n-1)(2n-3)$ fatal quadruple failures. We need to substract one from that product not to count twice the failure of drives D_{n-1}, P_{n-1}, D_n and P_n.
3. Any of the $n-2$ type C failures involving a data drive D_i, a data drive D_{i+2}, and the parity drives P_i and P_{i+1}.

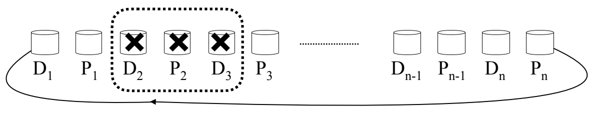
Hence, the probability γ that the entanglement will not tolerate a quadruple drive failure is:

$$\gamma = \frac{\binom{2n-2}{2} + (n-1)(2n-3) - 1 + (n-2)}{\binom{2n}{4}}\,.$$

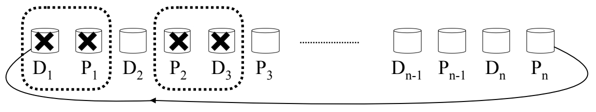
3.4.2 Closed Entanglements

Closed entanglements can tolerate all double drive failures without experiencing a data loss. As we can see on Figure 3.4, they share the same B and C irreducible failure patterns as open entanglements and have an additional irreducible triple failure that involves drives D_n, P_n and P_1. We will call this failure a *type D* failure.

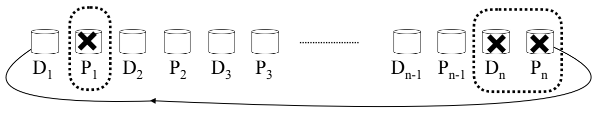
Given that closed entanglements tolerate all double failures without data loss, the probability α that the entanglement will not tolerate a double failure is zero.



(a) Closed entanglements - Type B failure



(b) Closed entanglements - Type C failure



(c) Closed entanglements - Type D failure

Figure 3.4 – The three irreducible fatal failure patterns of closed entanglements

The triple failures that will result in a data loss include:

- Any of the same $n-1$ type B triple failures as open entanglements.
- One type D triple failure.

As a result, the probability β that the entanglement will not tolerate a triple drive failure is:

$$\beta = \frac{n}{\binom{2n}{3}} \cdot$$

In the same way, the quadruple failures that will result in a data loss are:

- Any of the $n-1$ type B failures combined with the failure of any of the remaining $2n-3$ drives.
- The single type D failure combined with the failure of any of the remaining $2n-3$ drives.
- Any of the $n-2$ type C failures involving a data drive D_i, a data drive D_{i+2}, and the parity drives P_i and P_{i+1}.

The probability γ that the entanglement will not tolerate a quadruple drive failure is:

$$\gamma = \frac{n(2n-3)+(n-2)}{\binom{2n}{4}} \cdot$$

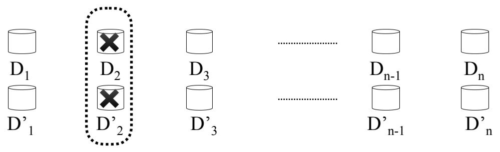


Figure 3.5 – The sole irreducible failure pattern of mirrored organizations

3.4.3 Mirrored Organizations

As we can see on Figure 3.5, the sole irreducible failure pattern for mirrored organizations is the failure of a mirrored pair. As a result, the probability α that a mirrored organization with $2n$ drives will not tolerate a double drive failure is:

$$\alpha = \frac{n}{\binom{2n}{2}} \cdot$$

The triple failures that will result in a data loss is the failure of any of its n mirrored pairs combined with the failure of any of the $2n-2$ remaining drives. Hence, the probability β that the mirrored organization will not tolerate a triple drive failure is:

$$\beta = \frac{n(n-2)}{\binom{2n}{3}} \cdot$$

In the same way, the number of quadruple failures that will result in a data loss comprises the failure of any of its n mirrored pairs combined with the failure of two of the $2n-2$ remaining drives. We must however subtract from this total the $\binom{n}{2}$ quadruple failures consisting of the failure of two mirrored pairs of drives in order not to count them twice. As a result, the probability γ that the organization will not tolerate a quadruple drive failure is:

$$\gamma = \frac{n\binom{2n-2}{2} - \binom{n}{2}}{\binom{2n}{4}} \cdot$$

3.5 Reliability Analysis

Estimating the reliability of a storage system means estimating the probability $R(t)$ that the system will operate correctly over the time interval $[0, t]$ given that it operated correctly at time $t = 0$. Computing that function requires solving a system of linear differential equations, a task that becomes quickly intractable as the complexity of the system grows. A simpler option is to use instead the five-year reliability of the array. As this value is typically very close to 1, we will express it in "nines" using the equation $number\_of\_nines = -\log_{10}(1 - R_d)$, where R_d is the five-year reliability of the array. Thus, a reliability of 99.9 percent would be represented by three nines, a reliability of 99.99 percent by four nines, and so on.

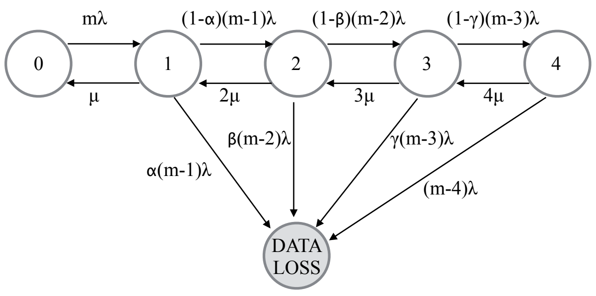


Figure 3.6 – State transition probability diagram

Array type	α	β	γ
Open entanglements	$\frac{1}{\binom{2n}{2}}$	$\frac{3n-3}{\binom{2n}{3}}$	$\frac{\binom{2n-2}{2}+(n-1)(2n-3)-1+(n-2)}{\binom{2n}{4}}$
Closed entanglements	0	$\frac{n}{\binom{2n}{3}}$	$\frac{n(2n-3)+(n-2)}{\binom{2n}{4}}$
Mirrored organizations	$\frac{n}{\binom{2n}{2}}$	$\frac{n(n-2)}{\binom{2n}{3}}$	$\frac{n\binom{2n-2}{2}-\binom{n}{2}}{\binom{2n}{4}}$

Table 3.1 – Probabilities α, β and γ that an entangled/mirrored array will not be able to tolerate double, triple and quadruple failures respectively.

We develop first a generic Markov model that will apply to open entanglements, closed entanglements and mirrored organizations. The specific behavior of each fault-tolerant drive array will be represented by the four parameters m, α, β, and γ, where $m = 2n$ is the number of disks in the array and α, β, and γ are the respective probabilities that the array will not tolerate the simultaneous failures of two, three or four drives. These values were calculated in the previous section and they are summarized in Table 3.1. In all three cases, we will neglect the probability that the array will tolerate a quintuple drive failure, assuming that this probability is small enough to be neglected.

The model consists of an array of drives with independent failure modes. Whenever a drive fails, a repair process is immediately initiated for that drive. Should several drives fail, this repair process will be performed in parallel on those drives. We assume that drive failures are independent events and are exponentially distributed with mean λ. In addition, we require repairs to be exponentially distributed with mean μ. Both hypotheses are necessary to represent our system by a Markov process with a finite number of states.

Figure 3.6 displays the state transition probability diagram. State <0> is the initial state where all m drives are operational and no drive has failed. Should any of the drives fail, the system would move to state <1> with an aggregate failure rate $m\lambda$. Since some double failures can be fatal, the two possible failure transitions from state <1> are:

1. A transition to the data loss state with rate $\alpha(m-1)\lambda$ where the actual value of the α parameter will depend on the specific storage organization, as computed in previous section.

2. A transition to state <2> with rate $(1-\alpha)(m-1)\lambda$.

In the same way, the two failure transition from state <2> are:

1. A transition to the data loss state with rate $\beta(m-2)\lambda$ where the actual value of the β parameter will depend on the specific storage organization.

2. A transition to state <3> with rate $(1-\beta)(m-2)\lambda$.

Following the same pattern, the two failure transition from state <3> are:

1. A transition to the data loss state with rate $\gamma(m-3)\lambda$ where the actual value of the γ parameter will depend on the specific storage organization.

2. A transition to state <4> with rate $(1-\gamma)(m-3)\lambda$.

As we did not take into account the possibility that the array could survive a quintuple failure, there is a single failure transition leaving state <4>.

Recovery transitions are more straightforward: they bring the array from state <4> to state <3>, then from state <3> to state <2> and so on until the system returns to its initial state <0>.

The Kolmogorov system of differential equations that describes the behavior of the storage organization is:

$$\begin{aligned}
\frac{dp_0(t)}{dt} &= -m\lambda p_0(t) + \mu p_1(t) \\
\frac{dp_1(t)}{dt} &= -((m-1)\lambda + \mu)p_1(t) + m\lambda p_0(t) + 2\mu p_2(t) \\
\frac{dp_2(t)}{dt} &= -((m-2)\lambda + 2\mu)p_2(t) + (1-\alpha)(m-1)\lambda p_1(t) + 3\mu p_3(t) \\
\frac{dp_4(t)}{dt} &= -((m-4)\lambda + 4\mu)p_4(t) + (1-\gamma)(m-3)\lambda p_3(t),
\end{aligned}$$

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

Observing that the mean time to data loss (MTTDL) of the system is given by:

$$\text{MTTDL} = \sum_{i=0}^{4} p_i^*(0)\,,$$

where $p_i^*(s)$ is the Laplace transform of $p_i(t)$, we compute the Laplace transforms of the above equations and we solve them for $s = 0$ and a fixed value of m. We then use this result to compute the MTTDL of our system and convert this MTTDL into a five-year reliability, using the equation:

$$R_d = \exp \frac{d}{\text{MTTDL}}, \tag{3.5}$$

where d is a five-year interval expressed in the same units as the MTTDL. Observe that the above equation implicitly assumes that long-term failure rate $1/\text{MTTDL}$ does not significantly differ from the average failure rate over the first five years of the array.

We analyzed the reliability of both open and close entanglements and compared them with the reliability of mirrored solutions under four different array configurations, namely:

1. A small array of 20 drives and a drive MTTF of 200,000 hours.
2. The same array with a drive MTTF of 35,000 hours.
3. A medium size array of 50 drives and a drive MTTF of 200,000 hours.
4. The same array with a drive MTTF of 35,000 hours.

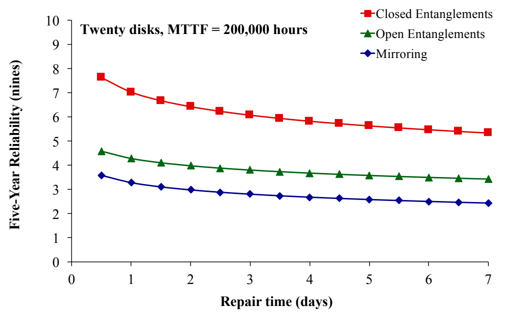
A drive MTTF of 200,000 hours, corresponds to an annual failure rate of 4.28 percent and represents what can be expected from an array built with very good drives. The annual failure rate is calculated using the inverse of the MTTF expressed in years. A drive MTTF of 35,000 hours corresponds to an annual failure rate of 25 percent. While this failure is pathological, it is neither exceptional nor confined to disks of dubious origin. Beach (Beach, 2014) reported just such a rate for a batch of 539 disks coming from a reputable manufacturer.

Figure 3.7 summarizes the findings. It shows the five-year reliabilities of the four configurations described above for mean time to repair (MTTR) varying between half a day and one week. Since reliabilities are expressed in nines, each unit increment on the vertical scale corresponds to a 90 percent reduction of the probability of a data loss.

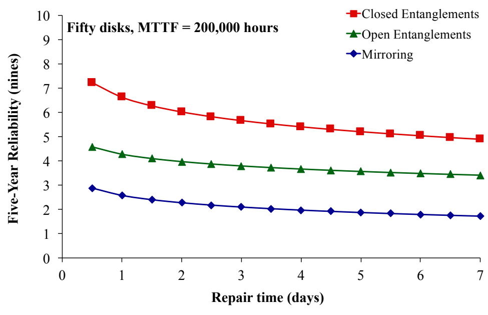
As we can see, open entanglements provide much better five-year reliabilities than mirroring and reduce the probability of a data loss by 90 percent for the small array and by 98 percent for the large array. The very good performance of open entanglements for large arrays should not surprise us given that these entanglements only have a single fatal double failure.

As expected, closed entanglements perform even better and reduce the probability of a data loss:

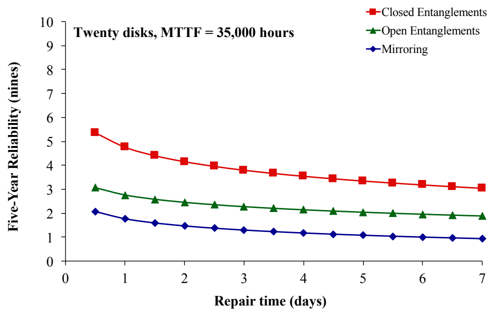
- by at least 99.87 percent for the small array and a 200,000-hour MTTF,
- by at least 99.93 percent for the large disk array and a 200,000-hour MTTF,



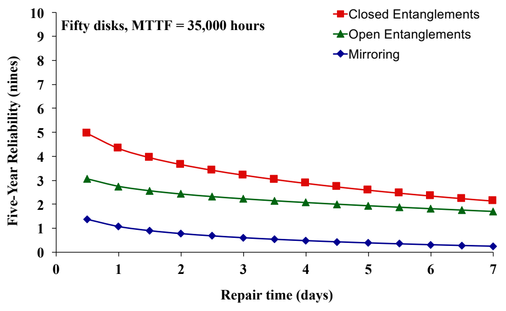
(a) 20-disk array with MTTF of 200,000 hours



(b) 50-disk array with MTTF of 200,000 hours



(c) 20-disk array with MTTF of 35,000 hours



(d) 50-disk array with MTTF of 35,000 hours

Figure 3.7 – Five-year reliability for different arrays

- by at least 99.21 percent for the small disk array and a 35,000-hour MTTF,

- by at least 98.68 percent for the large disk array and a 35,000-hour MTTF.

We should note that these are minimum values observed for very large drive repair times. The improvements observed for a more reasonable two day drive repair time vary between 99.79 and 99.98 percent.

Two features of our model may also affect the accuracy of our results. First, we assumed that drive failures were independent events, which is not always true. Second, we assumed that all quintuple drive failures were fatal. This assumption is fairly reasonable as long as the quintuple disk failures remain rare events, which is certainly true for small arrays consisting of drives with high MTTFs and low MTTRs. This is less true for larger arrays especially if their drives have lower MTTFs and higher MTTRs. As these arrays will experience more drive failures and have their failed drives remain for longer periods of time in that state, quintuple drive failures will become less uncommon. Assuming that all these failures are fatal will then provide pessimistic evaluations of the array five-year reliability.

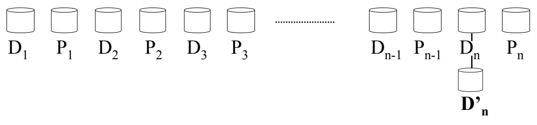


Figure 3.8 – An open entanglement that tolerates all double failures

3.6 Possible Extensions

While both open and closed entanglements provide much higher five-year array reliabilities than mirroring, there are circumstances where even higher levels of data protection must be sought. This could be the case if the array includes disk drives that exhibit high failure rates or if geographic considerations result in longer repair times. Let us show how we can address that issue by eliminating all fatal double drive failures in the case of open entanglements and all fatal triple drive failures in the case of their closed counterparts.

3.6.1 Open Entanglements

As we have seen in Section 3.4, an open entanglement with n data drives and n parity drives will not tolerate the combined failure of its last data drive D_n and its associated parity drive, drive P_n.

As we can see on Figure 3.8, the simplest solution is to mirror data drive D_n, thus adding an additional data drive D'_n. The outcome will be an array with $2n+1$ drives that would tolerate double drive failures and provide the same reliability as a closed entanglement with $2n$ drives.

3.6.2 Closed Entanglements

Recall that the sole fatal triple failures in closed entanglements are (a) type B triple failures involving data drive D_k, its associated parity drive P_k, and the next data drive D_{k+1} and (b) a single type D triple failure involving D_n, P_n and P_1. So, if we number sequentially all data drives starting with D_1, all type B triple fatal failures will involve both an odd-numbered and an even-numbered data drive. Thus, any mechanism that will allow the recovery of any failed odd-numbered data drive will eliminate all fatal type B triple failures.

To achieve this goal, we group all odd-numbered data drives into pairs (D_1, D_3), (D_5, D_7) and add to each pair an extra parity drive Q_j such that:

$$Q_j = D_{4k+1} \oplus D_{4k+3}\,,$$

where $0 \le k \le \lfloor \frac{n-3}{4} \rfloor$ and n is the number of data disks in the entanglement. In half of the cases, the pairing process will leave the last odd-numbered data drive alone. Then, two cases need to be considered:

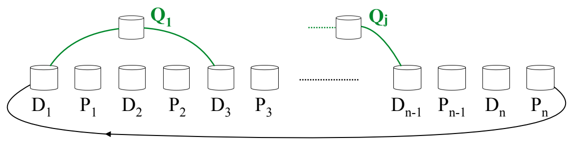


Figure 3.9 – A closed entanglement that tolerates all triple failures

1. If n is odd, the last odd-numbered data drive is drive D_n. We do not need extra protection since the drive is already entangled with the first drive of the chain, namely drive D_1.
2. If n is even, the last odd-numbered data drive is drive D_{n-1}. To protect data against the simultaneous failure of drives D_{n-2}, P_{n-2} and D_{n-1} and parity drive P_{n-2}, we must mirror drive D_{n-1} and have

   $$Q_x = D_{n-1}\,.$$

Figure 3.9 illustrates this last case.

3.7 Summary

We have investigated the reliability of two varieties of simple entanglements, namely open entanglements and closed entanglements. Both variants use equal numbers of data and parity drives and generate their parity data by computing the exclusive-or (XOR) of the most recently appended data with the contents of their last parity drive. Our study reveals that open and closed entanglements provide better five-year reliability than mirroring, reducing the probability of data loss by respectively 90 and 98 percent. Furthermore, these techniques are very efficient and simple to implement, either in software or hardware, and are hence of practical interest for cloud storage systems.

4 Complex Entanglements

The main contribution of this chapter is the introduction of *alpha entanglement codes*, a mechanism to propagate redundant information across a large scale distributed system and create a virtual storage layer of highly interconnected elements. Our solution mitigates the challenges of constructing a practical erasure code to archive big data. By "practical", we mean having reasonable trade-offs between security, resource usage and performance. The code has three parameters: one increases storage overhead linearly but increases the possible paths to recover data exponentially. Two other parameters increase fault-tolerance even further without the need of additional storage. As a result, entanglement codes can be used to design a robust redundancy scheme for a survivable storage system that would provide high availability, durability and integrity despite external and internal threats with little maintenance. Given the complexity of the topic, this chapter presents the model, analysis and simulations to understand the code behaviour to some extend and the next chapter presents the evaluation on reliability and fault tolerance. This chapter compiles and expands the following publications: Helical entanglement codes was published in SSS'13 (Estrada-Galinanes and Felber, 2013), our recent work alpha entanglement codes was submitted for publication (Estrada-Galinanes et al., 2017b) and the research framework to do simulations that will appear as extended abstract in SYSTOR'17 (Estrada-Galinanes, 2017).

4.1 Introduction

Our work Helical Entanglement Codes (HEC) (Estrada-Galinanes and Felber, 2013) introduces a technique that propagates redundant information across a storage system and provides multiple paths to recover unavailable content. HEC embrace two ideas: the creation of inter-dependencies among all content in a storage system (data entanglement) and the propagation of implicit redundant information using helical chains of data and parity blocks. The goal is to mix previously stored parity blocks with new data blocks to accumulate redundant information from previous blocks. The design combines multiple strands forming a helical lattice, hence, it tolerates more failures than simple entanglements. The entanglement computes the exclusive-or (XOR) of two blocks: a parity and a data block. Every new inserted block is entangled with three parities. The three outputs contribute to enlarge three strands and to propagate redundant information. The pattern is repetitive, i.e. the new parity block is used together with a newcomer block as input for the next XOR operation. Our starting point was p-HEC, entanglements with two horizontal strands and p double-helix strands that resemble DNA topology. HEC was later extended to alpha entanglement codes.

Alpha entanglement codes, $AE(\alpha, s, p)$, are a family of erasure codes built on data entanglement (Stubblefield and Wallach, 2001; Waldman and Mazieres, 2001; Aspnes et al., 2007; Estrada-Galinanes and Felber, 2013). They propagate redundant information in more directions than existing codes. The parameter α indicates the number of redundant blocks created per useful information (non-redundant blocks). Tuning the parameter α increases the possible paths to recover data exponentially while increasing the storage overhead linearly. They are classified in α-entanglement families: single entanglements ($\alpha = 1$) compute 1 parity per data block, double entanglements ($\alpha = 2$) compute 2 parities, triple entanglements ($\alpha = 3$) compute 3 parities and so on. For example, a triple entanglement with $s = 2$ and $p = 5$ is equivalent to the 5-HEC code described in our preliminary work (Estrada-Galinanes and Felber, 2013). The parameters s and p indicate the number of horizontal and helical strands that define a helical lattice. Tuning "s" or "p" does not modify the storage overhead generated by the encoder. In other words, these two parameters can increase fault-tolerance even further, without the need of additional parities.

This chapter presents the modelling, simulation and assessment of complex entanglement codes presented in three works (Estrada-Galinanes and Felber, 2013; Estrada-Galinanes and Felber, 2015b; Estrada-Galinanes et al., 2017b). Complex entanglements are generalized by alpha entanglement codes. Overall, this chapter focuses on double and triple entanglements, and it gives some ideas for future research on n-tuple entanglements. A deeper evaluation to determine the fault-tolerance of complex entanglements is presented in the next chapter.

4.2 Helical Entanglement Codes

The use of document entanglement was proposed for censorship resistant systems. We exploit here this technique to generate and propagate redundant information across storage devices. In preliminary work, we proposed helical entanglement codes (HEC) (Estrada-Galinanes and Felber, 2013). The entanglement algorithm creates interdependencies between old, current, and future content with only three operations. It propagates redundant information in regular patterns forming a lattice, as it is shown in Figure 4.1 and explained in more detail in Section 4.3.5. Figure 4.2 is a snapshot of the relationships within the connectivity graph that focuses on data block (node) "25". The algorithm keeps building connections after more data blocks are ingested by the system. The figure shows how three open entanglement chains like the ones shown in Figure 3.1a are intertwined to create the lattice. For simplicity, it highlights the connections of data block 25, but any data block participates in three chains. Although this visualization does not show the whole picture, it helps to understand the information flow

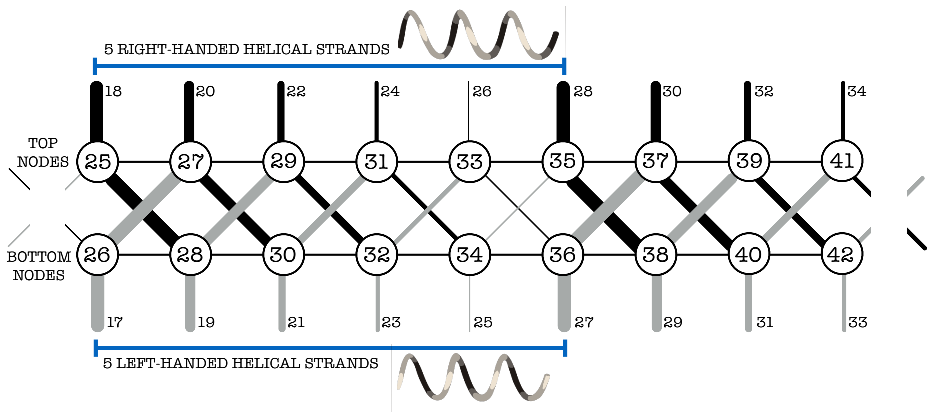


Figure 4.1 – HEC lattice. This graph shows how data blocks dependencies build a helical lattice. Nodes represent data blocks and edges represent parity blocks. Each helical strand is shown with a different thickness and colour.

from one node to another. More details about helical entanglement codes and its DNA-like appearance is presented in the next subsection.

4.2.1 DNA-like codes

The structure that models the relationship between data blocks resembles DNA helical strands; thus, we use the term helical and horizontal strands to refer to the entanglement chains. Note that the normal organic forms of double-stranded DNA are both right-handed, but our codes combine multiple left- and right-handed helical (double-helix) strands. The idea of storing data in DNA chains to create DNA-based storage solutions is being investigated by various authors (Church et al., 2012; Goldman et al., 2013; Bornholt et al., 2016). Their approach consists in writing data on synthesized DNA chains, which is radically different to helical entanglement codes.

A simplified view of the helical lattice is shown in Figure 4.2. This graph focus on how a data block that was ingested by the system at a certain time is connected to old and future data. It also shows the parity blocks that are created by the entanglement process. In particular, when the data block 25 arrives, it is XORed with parity blocks C,A, and E generating the parity blocks D, B, and F, respectively. In this view, it is easier to observe an important characteristic of the entanglement process. The helical strands meet periodically on the same horizontal strand. In our example, they meet at node 25 and at node 35 again. The period corresponds to the number of distinct double-helix strands p used by the entanglement algorithm (see Figure 4.1). Consequently, the distance between meeting points is p data blocks. For that reason, p-HEC is another way of referring to helical entanglement codes. We will see in next chapter that increasing p has a positive impact on the fault-tolerance of the code.

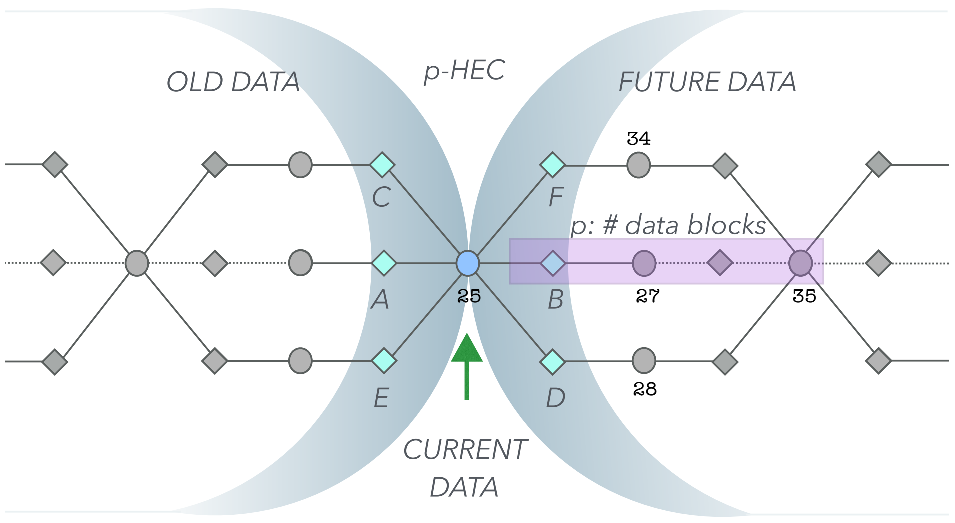


Figure 4.2 – DNA-like codes. Another view of the graph in Figure 4.1 showing the three strands used during the entanglement process (encoding) of one data block (25). The encoder uses parities C, A, and E to generate the parities D, B and F, respectively. This figure emphasizes the periodic behavior of helical strands that meet every p data blocks on the horizontal strand. For simplicity, only some data blocks (circles) have labels and parity blocks (diamonds) are explicitly shown.

4.2.2 Redundancy Propagation

A DNA-like entanglement graph has the positive effect of bounding the propagation of redundant information. The bound exists because the parities that are situated between the double-helix strand meeting points cannot be regenerated if all are lost and the data blocks at the meeting points are lost. A bound is necessary to protect all data blocks in a fair way. If propagation is not bounded, older blocks would increase their fault tolerance at the expense of slightly protected recent blocks. Additionally, the strands create parallel paths to recover data as they diverge and converge periodically. Therefore, fault tolerance is reinforced at a local level. Finally, the double-helix propagation occurs for all data blocks. In that sense, redundant information keeps propagating outside the bounds previously mentioned, hence increasing the number of failure patterns that the structure can tolerate.

4.2.3 Expanding Horizons

Later in this chapter, we present evidence of the higher survivability of entangled data in comparison with classical redundancy approaches. The DNA-like model was useful to define an efficient way of doing entanglements, however, it is not a general version for entanglement codes. It uses three strands and two horizontal strands. That triggers the question of what happens if we use a different number of strands? The single strand case defines the simple entanglements presented in previous chapter, but more studies were needed to explore different settings. Alpha entanglement codes are built on and improve our previous work to provide a general model for practical entanglement codes. In the next section, we define more families of entanglement codes and evaluate them in more detail.

4.3 Alpha Entanglement Codes

α-entanglements(s,p) is a family of erasure codes to tangle data and redundant blocks with the goal of increasing the scope of redundant information by means of propagation. The encoder builds block chains, *strands*, that alternate data and redundant blocks. The entanglement function computes the exclusive-or (XOR) of two consecutive blocks at the tail of a strand and inserts the output adjacent to the last block. The family of codes introduced in this work are:

1. *Single entanglements* ($\alpha = 1$), denoted by 1-entanglements, use a single horizontal strand. They are equivalent to the simple entanglements defined in Chapter 3.
2. *Double entanglements* ($\alpha = 2$), denoted by 2-entanglements(s,p), use horizontal strands and a single class of helical strand.
3. *Triple entanglements* ($\alpha = 3$), denoted by 3-entanglements(s,p), use horizontal strands and two classes of helical strands.
4. *n-Tuple entanglements* ($\alpha = n$), denoted by n-entanglements(s,p), use a horizontal strand and $n-1$ classes of helical strands.

A compact way to describe alpha entanglements is AE(α,s,p). The previous defined p-HEC method corresponds to the family of triple entanglements with $s = 2$, in short AE(3,2,p).

Figure 4.3 presents the fine- and coarse-grain view of α-entanglements. At the fine-grain level, the encoder and its parameter settings determine important attributes that are analyzed later in this chapter. At the coarse-grain level, the entanglement encoder algorithm creates deterministically the lattice connections that expand the potential use of redundant blocks.

This chapter focuses on double and triple entanglements, and it gives some ideas for future research on n-tuple entanglements.

4.3.1 Strands

Each individual strand corresponds to a single entanglement chain. The complex combination of strands brings additional properties and increases fault tolerance. One of the emergent properties of α-entanglements is that irreducible failure patterns from single entanglements become innocuous since data blocks are reconstructed through any of the α strands in which they participate.

There are three classes of strands, the horizontal (H), the right-handed (RH) and the left-handed (LH) strands. The last two are helical strands that take the name right-handed and left-

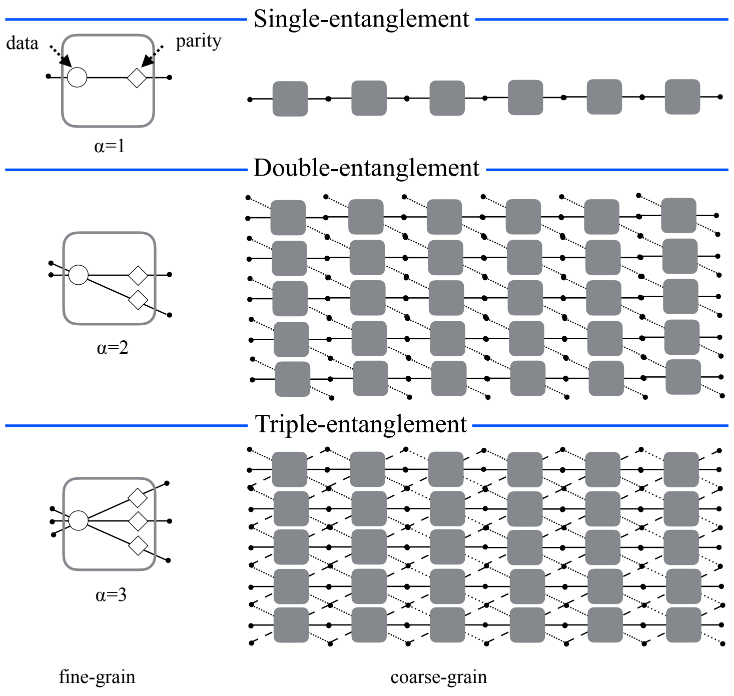


Figure 4.3 – Fine-grain and coarse-grain view of α*-entanglements.* The left side of the figure shows a bucket that contains a data block and the parities generated by the encoder. The right side shows how the buckets are interconnected. The connections between top and bottom buckets are not shown.

handed because the first encoder algorithm 5-HEC, i.e., a triple-entanglement with $s = 2$ and $p = 5$ describes a graph that resemble the illustrations of the DNA double-helix. Subsequent designs are better described with a helical lattice where the RH strands connect all H strands from top to bottom, with a certain slope, and jump to a subsequent position on the top H strand, and the LH strands define similar connections but going from bottom to top. The position after the jump is determined by the number of helical strands. In a 3D plane, the helical strands revolve around a central horizontal axis and grow towards the right direction.

A lattice is composed by s horizontal strands, and $\alpha - 1$ helical strand classes. Hence, the *total number of strands* is given by the expression

$$s + (\alpha - 1) \cdot p. \tag{4.1}$$

More details about the strands connectivity are given in the following subsections.

Slope

Finally, this work considers helical strands that connect horizontal strands with a diagonal of slope 1. In addition the number of RH and LH strands is balanced. In other words, 2-

entanglements are composed with one class of helical strands (RH or LH) and 3-entanglements are composed by both classes of strands. A lattice created with 3-entanglements with a single class of helical strands, for example, one RH strand with slope 1 and one RH strand with slope 2 is not studied.

4.3.2 Three Parameters to Propagate Information

Alpha entanglements expand the options available to propagate data by adding extra parameters.

Parameter α

This parameter quantifies the number of parities generated per each data block. It also describes the number of strands in which one data block participates.

Tuning α impacts on the resilience of the structure. More parities create more redundancy, on the other side, the storage overhead increases.

This work focuses on codes with $\alpha \in [1,3]$. We are still investigating entanglement codes with $\alpha > 3$. It is not clear how to connect the extra helical strands to form a regular mesh. In double- and triple- entanglements the RH- and LH-strands are defined along a diagonal of slope 1. One possible option is to add additional helical strands with a different slope.

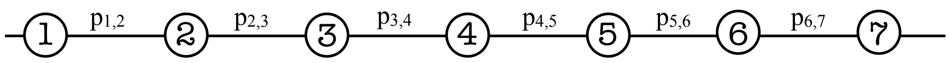
Parameters s and p

The total number of strands in the system is defined by the other two parameters: p, which determines the number of helical strands, and s, which defines the number of horizontal strands, for details see Figure 4.4.

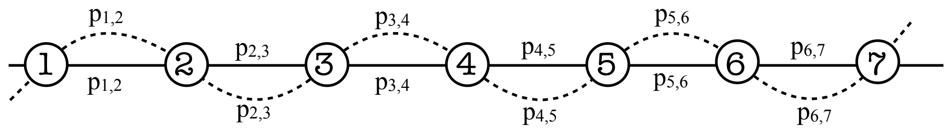
Tuning these two parameters impacts on the structure's resilience without adding storage overhead. A valid tuning parameter depends on α and the lattice topology.

- Single entanglements, namely 1-entanglement, are formed with only one chain of entanglements, therefore, $s = 1$ and $p = 0$.
- For entanglements with $\alpha > 1$, the only constraint to create a lattice is $p \geq s$. An invalid setting, i.e,. $p < s$, causes a deformed lattice.

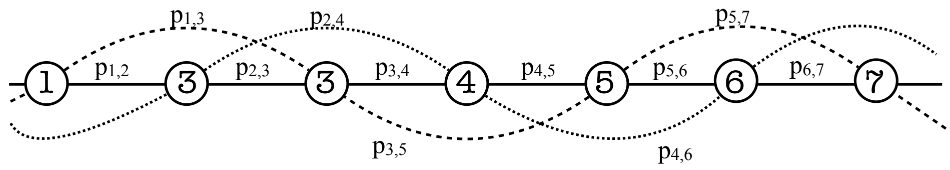
Figure 4.4 gives more insight into simple and double entanglements. Figure 4.4a is equivalent to Figure 3.1a for open entanglements. The examples for double entanglements show different code settings such as one horizontal strand ($s = 1$) as in Figures 4.4b, 4.4c or two horizontal strands as in Figure 4.4d. When $p = 1$ as in Figure 4.4b, the helical and horizontal strands are identical. Hence, the entanglement algorithm basically builds a simple entanglement chain



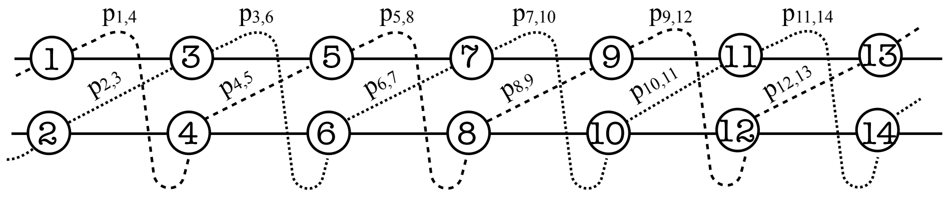
(a) 1-entanglements



(b) 2-entanglements(1,1)



(c) 2-entanglements(1,2)



(d) 2-entanglements(2,2)

Figure 4.4 – α*-entanglements* in more detail: Single and double entanglements with different settings.

and duplicates the parities. When $p \geq 2$ as in Figure 4.4d, each strand is built with different parities, which are computed by XORing different data blocks.

Our previous simplified view of the helical lattice shown in Figure 4.2 is now updated with the new parameters in Figure 4.5.

4.3.3 Code Rate

The code rate indicates the proportion between the amount of real information (data blocks) and the total amount of blocks. When all data blocks are stored in the system, the *code rate* is computed with

$$\frac{1}{\alpha+1}, \tag{4.2}$$

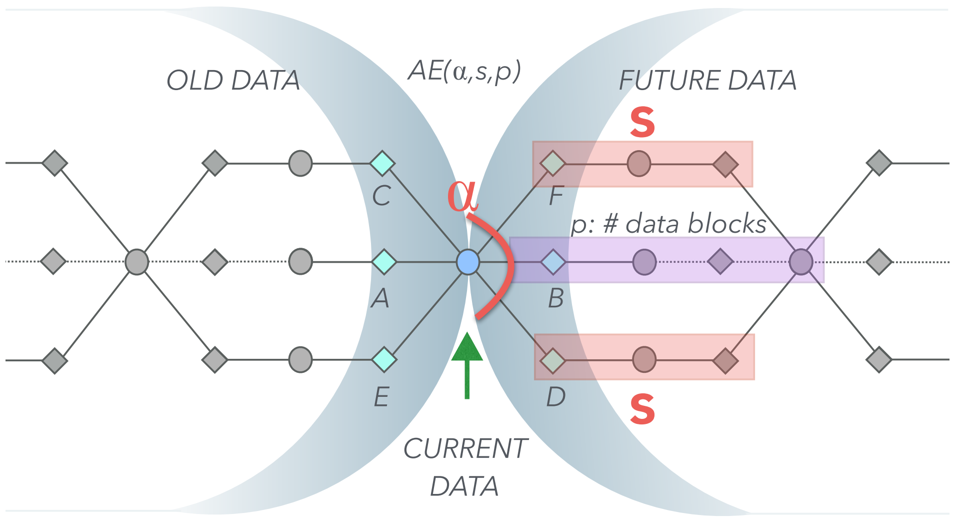


Figure 4.5 – Generalized version of the DNA-like code. Alpha entanglement codes incorporate more parameters to tune data propagation.

and the expression for systems than do not store data blocks is

$$\frac{1}{\alpha}\,. \tag{4.3}$$

Different approaches to increase the code rate of entanglement codes are investigated in Chapter 6.

4.3.4 Storage Overhead

For some applications, the storage overhead caused by α entanglements might be high. We propose three strategies to enhanced the code ratio. A first option is to start with a low α and increase the value later as required. A second option, is to puncture the code. Puncturing is a standard technique used in coding theory in which, after encoding, some of the parities are not stored in the system. Our study on puncturing and other XOR-based techniques are presented in Chapter 6. Finally, a third option could be to replace the XOR-based entanglement function for a more compact solution like Reed-Solomon to build “fat” data nodes. In our model, a data node is equivalent to one data block and all (data and parities) blocks have the same size. Such “fat” node would aggregate many data blocks and increase the code rate since in the same lattice region, more data blocks can be encoded. Our preliminary studies showed that a change of the entanglement function (fine-grain entanglement) has a negative impact in the size of failure patterns that cause data loss. To make this option possible, further analysis at the coarse-grain entanglement level is needed.

4.3.5 Helical Lattice

A graph *G(V,E)* consisting of a set of vertices V and a set of edges E describes the connections created by the entanglement process. By vertex we mean a node d_i that represents a data block (d-block), whereas an edge $p_{i,j}$ is an abstraction of a parity block (p-block). The graph is regular with degree $d = 2 \cdot \alpha$. The edge $p_{i,j}$ connects data vertices d_i and d_j. The underlying parity block contains derived information of the data block d_i and other data blocks previously used in the lattice.

The set of vertices V is divided in three disjoint subsets,

$$V = V_t \cup V_c \cup V_b\,, \tag{4.4}$$

with V_t, the subset of top nodes, V_c, the subset of central nodes, and V_b, the subset of bottom nodes, defined as follows:

$$V_t = \{d_i \in V \mid i \equiv 1 \pmod s, s > 1\}\,, \tag{4.5}$$

$$V_c = \{d_i \in V \mid i \not\equiv 1 \pmod s, i \not\equiv 0 \pmod s\}\,, \tag{4.6}$$

$$V_b = \{d_i \in V \mid i \equiv 0 \pmod s, s > 1\}\,. \tag{4.7}$$

Note that for single entanglements, where $s = 1$, all nodes are considered central.

A flat representation of this graph resembles a lattice. This two-dimensional model has a predefined height determined by the parameter s, i.e., the number of horizontal strands. The lattice grows uniformly from top to bottom "rows" and left to right "columns". Using the horizontal strands as a reference, the first d-block is added to H_1 (top), the second is added to H_2 (central), and so on, until H_s (bottom). When more d-blocks are added to the system, the sequence is repeated by adding them to the right column. Data blocks d_i are labeled with a unique identification i corresponding to the block's arrival order. As a result, for any d_i and d_j connected by the same horizontal strand, with $j > i$, the difference $j - i$ is a multiple of s.

Why is it called a helical lattice? The helical strands connect blocks from different rows and columns spanning an area defined by $s \cdot p$—the so-called *lead window*. A lead window describes the interval that takes one helical strand to revolve around an imaginary central axis. More details of a lead window are given in Section 7.3.4. Given a flat representation, once a right-handed helical strand connects a block from the bottom row, the next connection is located in the top row. Conversely, once a left-handed helical strand connects a block from the top row, the next connection is located in the bottom row. A three-dimensional model would have the shape of a cylinder. However, a simple two-dimensional model suffices to visualize the data entanglements. In the two-dimensional model, we relax the graph connectivities by leaving the top and bottom rows disconnected.

From the perspective of a storage system, the helical lattice is a virtual storage layer, which is somewhat alike to the one proposed in RESAR (Schwarz et al., 2016). The helical lattice

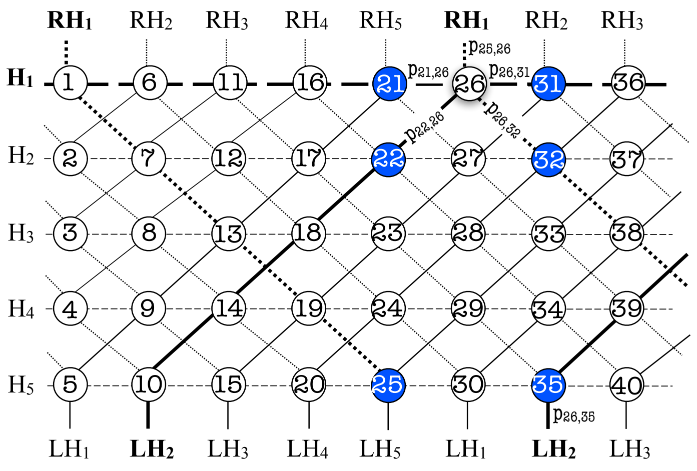


Figure 4.6 – Lattice for triple entanglements with $s = 5$ rows, and $p = 5$ columns or diagonals. A node d_i participates in three strands and, therefore, has degree $d = 6$. Colored vertices are at one hop of node d_{26}. d_{21} is a top node, d_{22} is a central node and d_{25} is a bottom node.

models the redundant propagation and the possible paths to recover a data chunk from a storage system. Noted that a lattice only makes sense for entanglements with $\alpha > 1$, since simple entanglements build a single chain.

Spanning Subgraphs

The set of edges in G is divided in α disjoint subsets,

$$E = E_1 \cup E_2 \cup \cdots E_\alpha, \tag{4.8}$$

where E_1 is the sole set of horizontal edges, also denote as E_H, and E_i with $i \in [2, \alpha]$ being a subset of helical edges.

Each of the α edge subsets form a spanning subgraph $S_i(V, E_i)$, whose set of vertices contains all the vertices of the original graph G. Due to the way information is propagated through each disjoint subset, this property allows the distribution of redundant information in α independent *availability zones* to tolerate network partitions.

4.3.6 The Family of Triple Entanglements

Figure 4.6 shows a helical lattice for triple entanglements with s=5 rows and p=5 columns or diagonals. A data node d_i participates in three strands and, therefore, has degree $d = 6$. Consequently, each node has six neighbor nodes. For example, colored vertices d_{21}, d_{22}, d_{25}, d_{31}, d_{32}, d_{35} are at one hop of node d_{26}. Nodes are classified according to their location in the lattice, for example: d_{21} is a top node, d_{22} is a central node and d_{25} is a bottom node.

	d_i is tangled with $p_{h,i}$, and h index is		
node d_i location	**H strand**	**RH strand**	**LH strand**
top	$i-s$	$i-s\cdot p+(s^2-1)$	$i-(s-1)$
central	$i-s$	$i-(s+1)$	$i-(s-1)$
bottom	$i-s$	$i-(s+1)$	$i-s\cdot p+(s-1)^2$

Table 4.1 – Entanglement rules to determine the triple entanglement's *input*.

	d_i entanglement creates $p_{i,j}$, and j index is		
node d_i location	**H strand**	**RH strand**	**LH strand**
top	$i+s$	$i+s+1$	$i+s\cdot p-(s-1)^2$
central	$i+s$	$i+s+1$	$i+s-1$
bottom	$i+s$	$i+s\cdot p-(s^2-1)$	$i+s-1$

Table 4.2 – Entanglement rules to determine the triple entanglement's *output*.

According to Equation 4.1, the total number of strands that build this lattice is 15. The set of edges is composed by three subsets (see Equation), $E = E_H \cup E_{RH} \cup E_{LH}$, labelled horizontal, right-handed and left-handed, respectively. Additionally each subset can be split in more disjoints subsets according to the parameter s and p. In the figure, each E_i contains 5 subsets, e.g., $E_H = E_{H_1} \cup E_{H_2} \cup E_{H_3} \cup E_{H_4} \cup E_{H_5}$

Given a node d_i, the incident edges, and therefore, the nodes that are at one hop of d_i, are predetermined by the rules specified in Tables 4.1 and 4.2 explained in Section 4.4.

Algorithm 4.1 shows a Prolog code snippet that uses the arithmetic rules given in Tables 4.1 and 4.2 to check connections in triple entanglement lattices. The same code can be used to prove connections for double entanglements if one of two helical strands is removed. It assumes that vertices in the grid are labeled with consecutive numbers, in vertical order and from left to right, starting with the node d_1 on the top left corner. For example, the vertex adjacent to d_1 in the first row and second column is d_6.

The code can be used to check the correct incident edges to d_i for a given lattice. As an example, we get the list of all incident edges to the top node d_{26} highlighted in Figure 4.6 using:

```
?- findall(Y, edge(5, 5, 26, Y, T), L).
L = [31, 21, 32, 25, 35, 22].
```

In the command above, we use the Prolog function `findall()` to list the index x for all possible $p_{x,26}$ and $p_{26,x}$ for the AE(3,5,5) lattice. When $x < 26$, the index goes at the left as in $p_{21,26}$, $p_{22,26}$ and $p_{25,26}$. When $x > 26$, the index goes at the right as in $p_{26,31}$, $p_{26,32}$ and $p_{26,35}$. The rule edge requires two parameters S and P to indicate the number of lattice rows (5) and columns (5) respectively. The third parameter is the node's index 26. These first three parameters must be specified as in the example given above. The fourth parameter Y corresponds to the missing index of a parity $p_{26,Y}$ or $p_{Y,26}$. The fifth parameter T indicates the strand and the position of the missing index. Since there are three strands, and two possible positions for the index (left and right), there are six valid atoms *{h, hi, rh, rhi, lh, lhi}*, where *h*, *rh* and *lh* indicate the

Algorithm 4.1 — Check Lattice Connections

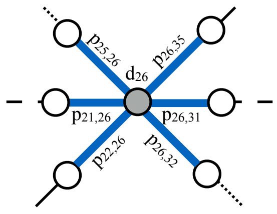
```

top(X, S) :− mod(X, S) =:= 1, S>1.
bottom(X, S) :− mod(X, S) =:= 0, S>1.
central(X, S) :− mod(X,S) > 1.

edge(S, _, X, Y, h) :− Y is X+S.
edge(S, _, X, Y, hi) :− Y is X−S.

edge(S, P, X, Y, rh) :− Y is X+S+1, (top(X,S); central(X,S))
        ; Y is X + S ∗ P − S^2 + 1, bottom(X, S)
        ; Y is X + P, S =:= 1
        .
edge(S, P, X, Y, rhi) :− Y is X−(S ∗ P−S^2+1), top(X,S)
        ; Y is X − (S+1), (central(X, S); bottom(X,S))
        ; Y is X − P, S =:= 1
        .
edge(S, P, X, Y, lh) :− Y is X+S ∗ P−(S−1)^2, top(X,S)
        ; Y is X + S−1, (bottom(X,S); central(X,S))
        ; Y is X + P, S =:= 1
        .
edge(S, P, X, Y, lhi) :− Y is X−(S ∗ P−(S−1)^2), bottom(X,S)
        ; Y is X − (S−1), (top(X,S); central(X,S))
        ; Y is X + P, S =:= 1
        .
```

strand and *i* indicates ”inverse”. The last three atoms are used to search from the missing index towards the beginning of the lattice. For instance, edge(5,5,26,Y,h) outputs 31 for edge $p_{26,31}$ but edge(5,5,26,Y,hi) outputs 21 for edge $p_{21,26}$. The function findall() will replace *T* by all valid atoms and list all possible values Y in the output L. Checking inexistent edges give false.

Due to the arithmetics involved, this code snippet gives invalid answers for nodes very close to the beginning of the grid. This limitation does not affect the edge validation. In fact, the entanglement algorithm uses fictitious parity blocks for all $p_{0,i}$. These fictitious parity blocks contain zeros and they are used when there are not enough blocks in the system.

4.3.7 Analogy with Other Codes

Alpha entanglement codes encode each data block α times using different parity blocks. Although different, they have similarities with coding schemes that code new information with information already coded. In serial concatenated codes, a series of codes are applied one after the other. In convolutional codes with feedback, the output of the shift registers is fed back and mixed with the new inputs. In staircase codes, coding progresses in a staircase fashion, and at each step new information is mixed with old information and old coded data to generate new coded data. Turbo codes and parallel concatenated codes encode the same data many times in parallel after interleaving it (Berrou et al., 1993; Berrou and Glavieux, 1996; Benedetto and Montorsi, 1996; Berrou and Glavieux, 2003). Alpha entanglement codes use



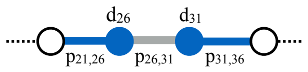


Figure 4.7 – Examples of entanglement tuples based on the lattice shown in Figure 4.6. The pp-tuples associated to d_{26} are $\{(p_{21,26}, p_{26,31}), (p_{25,26}, p_{26,32}), (p_{22,26}, p_{26,35})\}$. The dp-tuples associated to $p_{26,31}$ are $\{(d_{26}, p_{21,26}), (d_{31}, p_{31,36})\}$

.

parallel sub-coders, but each of them processes different subsets of data. For example, in parallel concatenated convolutional codes (PCCC) with feedback the encoder is applied to two permutations of the same input. In other words, if the input is “abacad” one sub-coder receives “abacad”, and the other would receive the permutation “acbaad”. Instead, in α-entanglements(s,p), the input “abacad” is distributed across s sub-coders and each sub-coder generates α outputs (and the systematic one). The next section illustrates how the setting 3-entanglement(2,5) works.

4.4 Encoder

The encoder computes α parity blocks per each data input. The core of the encoder algorithm for complex entanglements is not very different from simple entanglements. The equation to compute a parity is a generalized version of Equation 3.1 given in previous chapter,

$$p_{i,j} \leftarrow d_i \oplus p_{h,i}, \tag{4.9}$$

for i greater than zero. The values for indexes h and j are determined by entanglement rules.

Tables 4.1 and 4.2 determine the encoder's input and output, respectively. The tables have three columns, one for each strand, and three rows for top, central and bottom nodes. For example, they can be used to determine the input/output parities of the RH entanglement for d_{26} of Figure 4.6. According to the illustration, d_{26} belongs to RH_1 strand with incident parities $p_{25,26}$ and $p_{26,32}$. Table 4.1 determines the h index for parity $p_{h,26}$ as $h = 26-5\cdot5+(5^2-1) = 25$. Table 4.2 determines the j index for parity $p_{26,j}$ as $j = 26+5+1 = 32$. Note that single entanglements only use a small rule subset: central location and H strands rules. In this particular case, Equation 4.9 is identical to Equation 3.1. Double entanglements need only the rules for H strands, and RH strand or LH strand. Finally, triple entanglements make use of all rules.

The ultimate goal of the entanglement process is to create multiple tuples. An *entanglement tuple* is a two-element set that is used to re-create a data or a parity block. There are two types:

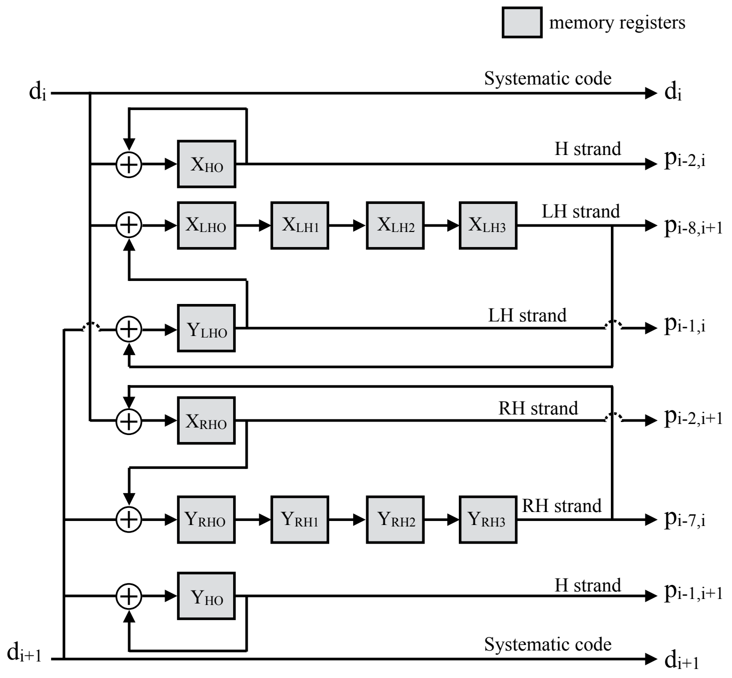


Figure 4.8 – A possible encoder for a helical lattice built with triple entanglements, $s = 2$ and $p = 5$.

pp-tuple	It is a 2-tuple composed by two consecutive parity blocks that are part of the same strand. Each data block has α associated pp-tuples, one for each strand class.
dp-tuple	It is a 2-tuple composed by one data block and one adjacent parity block on the same strand. Each parity block has 2 associated dp-tuples, both belonging to the same strand.

Figure 4.7 presents some examples.

The lattice design allows the computation of $\alpha \cdot s$ parities in parallel. Full-write column is possible since the helical lattice grows from top to bottom and from left to right; hence all the needed blocks are available to encode a column simultaneously. A column is composed with s data blocks d_i with $i \in [c+1, c+s]$ for some constant c multiple of s. For further optimization, the entanglement process needs to keep in memory $\alpha \cdot s$ parities, if $s = p$, these parities were calculated in the previous entanglement. More details are given in Section 7.3.6.

Encoder Example

Figure 4.8 presents an schematic design diagram for a 3-entanglement(2,5) encoder. This encoder could be implemented in software or hardware. The design illustrates the analogy to parallel concatenated convolutional codes given in Section 4.3.7. The figure shows two ($s = 2$) identical subcoders in parallel, each of them processing a distinct data block input.

Time	Input		Horizontal strand				Right-handed strand							Left-handed strand						
			H Registers		Outputs		RH Registers					Outputs		LH Registers					Outputs	
	d_i	d_{i+1}	X_{H0}	Y_{H0}	$p_{i-2,i}$	$p_{i-1,i+1}$	X_{RH0}	Y_{RH0}	Y_{RH1}	Y_{RH2}	Y_{RH3}	$p_{i-7,i}$	$p_{i-2,i+1}$	X_{LH0}	X_{LH1}	X_{LH2}	X_{LH3}	Y_{LH0}	$p_{i-1,i}$	$p_{i-8,i+1}$
-1	-	-	0	0	-	-	0	0	0	0	0	-	-	0	0	0	0	0	-	-
0	1	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
1	1	0	1+1	0	1	0	1	0+1	0	0	0	0	1	1	1	0	0	0	0	0
2	0	1	0	1	0	0	0	1+1	1	0	0	0	1	0	1	1	0	1	0	0
3	1	0	1	0+1	0	1	1	0	0	1	0	0	0	1+1	0	1	1	0	1	0
4	0	1	0+1	1+1	1	1	0	1+1	0	0	1	0	1	0	0	0	1	1+1	0	1
5	1	1	1+1	1	1	0	1+1	1	0	0	0	1	0	1	0	0	0	1+1	0	1
6	0	1	0	0+1	0	1	0	1	1	0	0	0	0	0	1	0	0	1	0	0
7	1	0	1	0	0	0	1	0	1	1	0	0	0	1+1	0	1	0	0	1	0
8	1	0	1+1	0	1	0	1	0+1	0	1	1	0	1	1	0	0	1	0	0	0
9	0	1	0	1	0	0	0+1	1+1	1	0	1	1	1	0	1	0	0	1+1	0	1
10	0	1	0	0+1	0	1	0	1	0	1	0	1	1	0	0	1	0	1	0	0

Table 4.3 – State of the registers and parity outputs ($p_{i,j}$) for the encoder shown in Figure 4.8. Some columns correspond to registers that store the result of XORing one of the inputs with one of the outputs. Since the output is fed-back into the system, a table value that has "+1" indicates the cases when the feedback line is positive, i.e. the output $p_{i,j} = 1$. For example, register $X_{H0} = d_i \oplus p_{i-2,i}$ has the value "1+1" at time 8 because $p_{i-2,i}(8) = 1$.

Data blocks are identified with a sequential index number i. The naming rule does not imply that d_i and d_{i+1} arrive at a different time. Both inputs are processed at the same time. Each of these sub-coders has three ($\alpha = 3$) pipelines (one per strand). In addition, there is a cross feedback between sub-coders. This cross-feedback is needed to construct the helical strands (RH and LH outputs). Basically, the input sequence is kept in register memories. In this case, 12 registers are needed, more precisely, 6 registers in each of the 2 (s) subcoders: a) one register for the horizontal strand, b) four ($p-1$) registers for the helical strand that revolves around a virtual central axis of the lattice and meet the same horizontal strand at a future time, and c) one additional register for the helical strand that generates the parity that is immediately used in the next step.

Table 4.3 shows the state of the memory registers and outputs corresponding to this encoder example. For the sake of simplicity, we assume that a block is 1-bit long. In real applications, block encoding, e.g., 1 MB, can be performed in parallel by multiple encoders. A clock controls the movement of data from one register to other. At each clock tick, the bits are shifted and combined to form the output parities. Initially, all registers contains 0. Their value is updated when new data arrives. In the example, the encoder process a finite amount of data, but the encoder can process data continuously.

The example shows how the stream *1010011001110110100101* is encoded. At each tick two digits are moved into the inputs (d_i and d_{i+1}). Equivalently, we could divide the message into two streams inputs: the one containing the odd bits is assigned to d_i and the second stream, which is formed with even bits, is assigned to d_{i+1}. Internally, the encoder uses the Equation 4.9 to generate the parities for each strand with each of the two inputs. Therefore, at each clock tick the encoder releases the six parities used in the right part of the equation. Since the code is systematic, the output lines include the inputs too. The output parities, at a certain time, were generated with data block inputs from different clock ticks. The actual block input is indicated in the left part of the output parity index, which was obtained with Table 4.1. The registers allow keeping the new parities (left part of the same equation) in memory until the proper clock tick enables re-encoding them with future inputs and releasing them. Some lines are formed with a serial combination of registers. The content of each register is transferred from left to right at each clock tick. In other words, in a serial combination of registers X_0 and X_1, the value of $X_1(t)$ is $X_0(t-1)$ with time t. Each line's output is fed back into the first register of the same or other line, which is preceded by a XOR gate. According to the truth table of XOR, the feedback can change the register value ony if it is "1". Table 4.3 indicates these cases by adding "+1" in all the first registers (X_{H0}, Y_{H0}, X_{RH0}, Y_{RH0}, X_{LH0}, Y_{LH0}). Finally, at the clock tick 10, the registers keep their values until the encoder processes the next coming stream.

4.5 Decoder

In the absence of faults, read requests are answered directly with the uncoded data. Decoding is an alternative way to obtain data through redundant sources with a twofold purpose: recover data in the presence of system faults and allow load-balanced reads.

This section presents three mechanisms for decoding data in the presence of system faults, namely local, zonal, and global repair. The size and characteristics of the event dictates which method is the appropriate one to use. The methods are valid for entanglements with $\alpha \geq 2$, except for local repair that applies to single entanglements too.

System faults can impact on multivariate elements of the helical lattice. These are the possible scenarios, roughly categorized into three groups:

1. *Small local damage* affects only a small region with a single data node, available or not, as its epicenter. It occurs when certain conditions are met: (a) the maximum number of missing incident edges is $\alpha - 1$, and (b) each missing edge has at least one of its associated dp-tuple available. Exceptionally, two incident edges that belong to the same strand for entanglements with $\alpha = 2$ are part of this category provided that the second condition is met. The restrictions assure that repair will success in one round and all elements can be repaired in parallel.

2. *Medium zonal damage* affects various consecutive elements on the same strand to a degree that decoding requires more than one round. Each missing data node must be part of a small local damage. It is called a zonal damage since this damage may impact on one availability zone as described in Section 4.3.5. Availability zones are implementation details discussed in Section 7.3.4.

3. *Large global damage* affects a wide area of close nodes and could be a combination of the other two damage categories. All erasure patterns that are not part of the previous categories and require more than one repair round fall in this category.

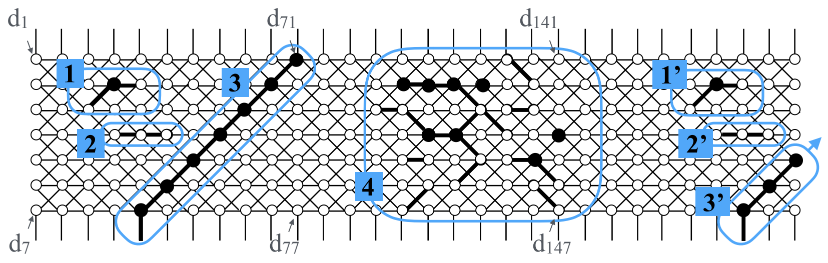


Figure 4.9 – System faults impact on the helical lattice and create different scenarios: regions 1 and 2 are small local damaged areas, region 3 is a medium zone damage, region 4 is a global damage. In the example, lattice's elements are mapped regularly to identical storage devices causing damages to appear periodically. The nodes d_1, d_7, d_{71}, d_{77}, d_{141}, and d_{147} are given as reference positions.

The scenarios are tentatively classified as small, medium and large regions, although, the number of elements in each scenario is variable. Examples for the three scenarios are illustrated in Figure 4.9. In addition, the figure shows periodical damages that occur if lattice's elements are mapped regularly to identical storage devices, i.e. a damaged region will appear at regular intervals in the structure.

4.5.1 Overview of the Repair Algorithms

The basic function of the repair algorithm is to recover the structure of the helical lattice by recreating the exact missing blocks. We assume that one or more repair agents trigger repair operations according to system policies, which are not discussed in this document. According to the scenarios described before, there are three repair approaches.

Local Repair

Local repairs can recover small local damaged areas, and other small erasure patterns, with a success rate of 100% as long as tuples associated with all missing elements are available. Successful repair means that all elements are recoverable. The expression used to re-create a data block is a general form of Equation 3.2, given in previous chapter for simple entanglements,

$$d_i \leftarrow p_{h,i} \oplus p_{i,j}\,. \tag{4.10}$$

Values for the parameters are taken from one of the α pp-tuples associated with the missing data node d_i.

The repair method to re-create a parity block uses Equation 4.9, with the values for parameters taken from one of the two dp-tuples associated with the missing parity.

The entanglement tuples are obtained with the rules defined in Tables 4.1 and 4.2.

Region	Nodes	Tuples	Edges	Tuples
1	d_{23}	$(p_{15,23}, p_{23,31})$	$p_{17,23}$	$(d_{17}, p_{11,17})$
	-		$p_{23,30}$	$(d_{30}, p_{30,37})$
2	-		$p_{25,32}$	$(d_{25}, p_{25,18})$
	-		$p_{32,39}$	$(d_{39}, p_{39,46})$
3	d_{35}	$(p_{28,35}, p_{35,42})$	$p_{22,35}$	$(d_{22}, p_{16,22})$
	d_{41}	$(p_{34,41}, p_{41,48})$	$p_{35,41}$	♦ $(d_{35}, p_{22,35})$
	d_{47}	$(p_{40,47}, p_{47,54})$	$p_{41,47}$	♦ $(d_{41}, p_{35,41})$
	d_{53}	$(p_{46,53}, p_{53,60})$	$p_{47,53}$	♦ $(d_{47}, p_{41,47})$
	d_{59}	$(p_{52,59}, p_{59,66})$	$p_{53,59}$	♦ $(d_{53}, p_{47,53})$
	d_{65}	$(p_{58,65}, p_{65,72})$	$p_{59,65}$	♦ $(d_{59}, p_{53,59})$
	d_{71}	$(p_{64,71}, p_{71,78})$	$p_{65,71}$	♦ $(d_{71}, p_{84,90})$

Table 4.4 – List of lost nodes and edges from regions 1-3 shown in Figure 4.9 and the elements used by the decoder algorithm (pp-tuple or dp-tuple). The symbol "♦" indicates that the tuple is not available in the first decoding round.

Zonal Repair

This method combines local repairs recursively to recover a medium zonal damage with a success rate of 100%. The total number of recursive calls is computable under the assumption that the repair algorithm works on a snapshot of the system, or that faults do not occur while the repair algorithm is running. It is at most $\lceil \frac{n}{2} \rceil + 1$, with n being the number of missing edges. Repairs occur at both extremities of the damaged strand section and proceed towards the centre. The maximum number of recursive calls is reached when the edges at the extremity do not have any dp-tuple available and one extra step is needed to repair data blocks.

Global Repair

This method combines local and zonal repairs to recover from large damages. A straightforward repair strategy is a *greedy algorithm.* The downside of greedy repairs is that the algorithm may repair first the nodes with less risk of permanent loss. If failures occur during repairs, the most vulnerable nodes might become irrecoverable. Then, if the system does not have the resources to fix all local repairs at once it may be safer to use a *local-wise strategy.* A local-wise strategy uses a *weight function* $W(d_i)$ that measures node risk to prioritize repairs. The simplest weight function calculates the number of missing incident edges. An improved weight function $W(d_i, h_o)$ will consider the states of neighbor nodes up to a certain number of hops h_o.

Global repairs have a success rate that ranges from 0 to 100%. The success rate is affected by the appearance of irreducible failure patterns. More information about irreducible failures is given in the next chapter.

Algorithm 4.2 — Global decoder with a greedy strategy.

```
1: $\mathscr{K}$: List of missing elements' keys $k_1, k_2, \dots$    ▷ $k$ is the index of a data node $d_k$ or an edge $p_k$
2: $\mathscr{R} \leftarrow \emptyset$    ▷ Create and initialize a sublist for elements that meet the conditions for local repairs
3: repeat
4:     for each $k \in \mathscr{K}$ do
5:         if IsLocalyRepairable($e$) then    ▷ Verify if conditions for local repairs are met
6:             $\mathscr{R} \leftarrow k$    ▷ Append the key $k$ to $\mathscr{R}$
7:         end if
8:     end for
9:     for each $k \in \mathscr{R}$ do
10:        if IsNode($k$) then
11:            $t_{PP} \leftarrow$ GetPP-Tuple($k$)    ▷ $t_{PP} = (p, p')$ is an available $pp$-tuple associated with $d_k$
12:            $d_k \leftarrow p \oplus p'$    ▷ $d_k$ is the recovered data block
13:        else
14:            $t_{DP} \leftarrow$ GetDP-Tuple($k$)    ▷ $t_{DP} = (d, p)$ is an available $dp$-tuple associated with $p_k$
15:            $p_k \leftarrow d \oplus p$    ▷ $p_k$ is the recovered parity block
16:        end if
17:    end for
18:    $\mathscr{R} \leftarrow \emptyset$
19: until IsEmpty($\mathscr{K}$)    ▷ If there are no missing elements go to sleep
```

4.5.2 Repair Details

Given the lattice in Figure 4.9 and assuming it corresponds to AE(3,7,7), we can describe how the repair methods work. Region 1 and 2 are small locally repairable regions, hence all tuples are listed as available. Region 3 is an example for a medium zonal damage. We know that, for region 3, the algorithm will be successful in at most 5 recursive calls. Region 4 is an example for a global damage. Table 4.4 enumerates the missing elements and, for each of them, one of their associated tuples to use during the repairs of regions 1 to 3.

The repair method can repair all missing elements from regions 1 and 2 indicated in Figure 4.9 simultaneously. Regions 1 has three missing elements, one data node d_{23} and two incident edges $p_{17,23}$ and $p_{23,30}$ from different strands. Region 2 has two missing elements, two edges $p_{25,32}$ and $p_{32,39}$ from the same strand. Both regions can be repaired in a single round using the available tuples indicated in Table 4.4.

Region 3 has fourteen missing elements, seven data nodes and seven edges from the same strand. The first repair round recovers all data nodes d_{35}, d_{41}, d_{47}, d_{53}, d_{59}, d_{65}, and d_{71} and the edge $p_{22,35}$. The second round recovers $p_{35,41}$ and $p_{65,71}$. The third round recovers $p_{41,47}$ and $p_{59,65}$. Finally, the fourth round recovers $p_{47,53}$ and $p_{53,59}$. If the damage area is not affected with more failures during repairs, then, the algorithm successes after four calls.

Region 4 is a larger area that comprises multiple damages and, therefore, it starts a *global repair* process. Algorithm 4.2 presents pseudocode to repair the missing elements in a greedy fashion. The first **for** loop at line 4 creates a list with all the elements that are locally repairable. The second **for** loop at line 9 proceeds to repair them. The process is repeated until the list of

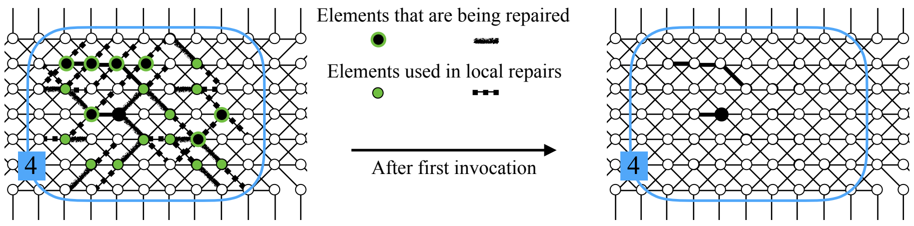

Elements that are being repaired
Elements used in local repairs
After first invocation
4
4

Figure 4.10 – Left picture identifies the elements involved in local repairs using a greedy strategy. Right picture shows the results after the first invocation.

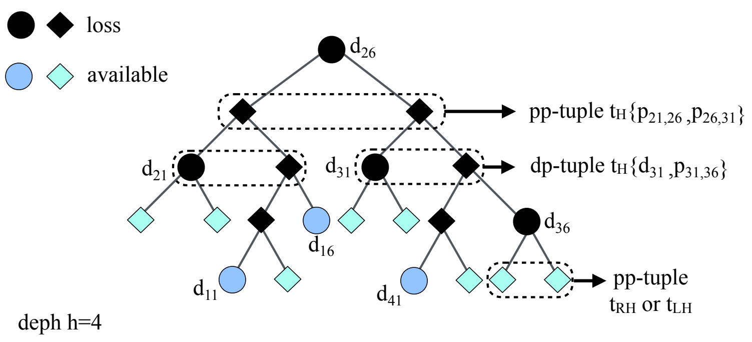


Figure 4.11 – A recovery tree upon global damage in a $AE(3,5,5)$ lattice. The root d_{26} is recovered if all leaves of the recovery tree are available (or can be recursively regenerated). For simplicity, only d-blocks have labels. This tree is initiated with pp-tuples from the H strand. Only at inner d-blocks nodes the strand can be changed.

missing elements becomes empty. Figure 4.10 illustrates how the greedy algorithm repairs most of the missing elements after the first invocation.

4.5.3 Reads During Failure Mode

What happens if the list of missing blocks is not know? This can happen for instance if a user is trying to retrieve one block from an unavailable location. Additionally, other locations could be unavailable without the user awareness. The greedy algorithm presented in Algorithm 4.2 is useful for recovering the system from failure mode but it requires knowledge of the blocks that are not available, hence, it is valid for internal system maintenance. However, global repairs may not be triggered especially if the locations are only temporarily unavailable. From the perspective of a user, it is important to recover a specific block even if the system is not working as expected.

The process of recovering a single block affected by a global damage is described by a *recovery tree*. The tree leverages the inherent redundancies in a helical lattice. The process is initiated when none of the α pp-tuples that facilitates a local repair are available or complete. In essence, each pp-tuple is recovered by repairing each of its constituent p-blocks, which are recovered by means of dp-tuples. Recursively, the components of dp-tuples are regenerated using the corresponding tuple.

Figure 4.11 illustrates a possible recovery tree for node d_{26}. The recovery tree is valid for the $AE(3,5,5)$ lattice shown in Figure 4.6. The root d_{26} is recovered if all leaves of the recovery tree are available (or can be recursively regenerated). The example requires a tree of *depth* $h = 4$ to read d_{26}. This tree can be initiated with pp-tuples from any of the three strands. Once a strand is chosen, the same strand is used to do repairs at different tree levels. Only at inner d-blocks nodes the strand changes. The reason for this is that p-blocks have only two ways of being recovered and both of them are defined on the same strand. Because d-blocks belong to α strands, each inner d-block diversifies the possible trees, creating multiple options. The parent node of an inner d-block is a missing p-block. That leaves $\alpha - 1$ pp-tuples options for each inner d-block.

4.6 Evaluation

This section presents preliminary experiments done for 5-HEC codes and simulations to compare alpha entanglement codes with (k,m)-codes.

4.6.1 Initial Assessment

The experiment consists of performing the actual encoding of a 700MB file with three different methods. The file is split into 700 blocks of 1 MB. The three encoders are: 5-HEC, 4-way replication and (4,12) RS. The 5-HEC encoder and the 4-way replicator generate three redundant blocks for each individual data block. The (4,12)RS encoder takes four data blocks and generates 12 redundant blocks. For all cases, the system has $700 \cdot 4$ total blocks. The purpose is to show the ability of different codes to recover data after a massive failure event.

A failure scenario is generated by choosing blocks at random and deleting all selected blocks. The percentage of deletions is gradually increased for each scenario. For the case of replication, data loss occurs when the four existent copies of one block are deleted. In the RS encoded file, data loss occurs when 13 blocks of a stripe of $4+12$ blocks are deleted. In the 5-HEC encoded file, data loss occurs when deleted blocks form an erasure pattern not tolerated by the lattice.

Table 4.5 shows the percentage of the file blocks that remain unrecoverable after trying to recover them with the remaining blocks. The results provide evidence of the strong positive impact on data durability. With 55% of deleted blocks in the 5-HEC structure, only a small percentage of data cannot be recovered. Data loss is due to the appearance of minimal erasure patterns with size 11. We tried different configurations of HEC and found that we could recover all data when a 200-HEC was used. A 200-HEC lattice generates the same number of encoded blocks per each data block as the 5-HEC lattice. The main difference is that one of the strands in which the data block participates will not grow until 400 ($2 \cdot 200$) blocks are added to the system. Because the number of blocks located at the end of the lattice are not

	Percentage of deleted blocks					
	5	**15**	**25**	**35**	**45**	**55**
5-HEC						
Data loss	0.00	0.00	0.00	0.00	0.00	2.41
4-way replication						
Data loss	0.00	0.05	0.25	1.72	4.41	9.26
Vulnerable data	0.05	1.27	5.20	10.78	20.29	31.08
(4,12) Reed-Solomon						
Data loss	0.00	0.00	0.00	0.10	0.39	3.19
Vulnerable data	0.00	0.00	0.00	0.15	0.88	14.56

Table 4.5 – Data loss and redundancy loss

fully entangled with other data blocks these blocks require some additional protection. In other words, a 200-HEC lattice increases the overhead generated for the last 400 blocks,for example by replicating them. Although 400 additional copies may seem a large number, it becomes insignificant when we consider a system with millions of blocks.

We also indicate the percentage of data that is vulnerable, i.e., that lost all redundant blocks. Data vulnerability is an interested metric that can be used to estimate the risk of data lost if there are more failures. This study was limited to classical codes as it was difficult to measure lost of redundancy in a system where all content is interdependent. Another limitation of this study is that placements are not considered, i.e., we assume that each block is stored in a distinct location. Finally, this study only considers 700 blocks. It is expected that even 200-HEC may lose some data if the system stores a very large number of blocks. In other words, it is more likely that a pattern that cause data loss will appear in a larger sample.

Our recent evaluations done in a simulated environment extend this study.

4.6.2 Simulations

We run experiments over millions of synthetically generated blocks without performing any actual encoding and/or decoding files. The simulations allow us to extend our previous study on data loss and vulnerable data in an efficient manner. This evaluation takes into account the distribution of blocks in a certain number of locations to investigate random placements. Each location could be a disk in a RAID-like system or a storage node in a peer-to-peer network. Locations may potentially fail or being temporarily unavailable. This experiments are conducted over many redundancy methods, which cause different percentange of storage overhead. Comparing a larger number of codes let us address the question of how much redundancy is needed to keep data safe in unreliable environments without requiring maintenance. Finally, we measure the impact of single failures for different redundancy schemes The tools used in this study are an ongoing effort that we plan to make available online (Estrada-Galinanes, 2017).

Examples of unreliable environments. An unreliable environment could be a peer-to-peer

Cost	RS(10,4)	RS(8,2)	RS(5,5)	RS(4,12)	AE(1,-,-)	AE(2,2,5)	AE(3,2,5)
Extra storage	40%	25%	100%	300%	100%	200%	300%
Single failures	10	8	5	4	2	2	2

Table 4.6 – Redundancy schemes.

Stripe	Block	Location	Available	Repaired
100	0	54	True	False
100	8	98	True	False

Table 4.7 – Table representation for code RS(8,2): Block values less than eight represent data blocks and values eight and nine represent parities.

(p2p) network where nodes join and leave frequently. Redundancy helps to protect data from real node departures and for maintaining high availability when hosts are offline. A relevant work on p2p storage showed a large-scale cooperative storage is limited by unreasonable cross-system bandwidth (Blake and Rodrigues, 2003). This study brings interesting insights from real world p2p networks, although some results need to be updated to current hardware trends and Internet connections. The authors confirmed that nodes availability in a p2p network is very variable. They add that maintenance redundancy consumes most of the node's resources. Another relevant remark from the same work is that for achieving high availability (6 nines) the system needs a replication factor of up to 120, while erasure codes require 15 times the storage. In archival storage systems, data durability is an endeavour that depends on the engineering aspects of the system but on the economics too (Adams et al., 2011; Rosenthal et al., 2012). To give a rough idea of maintenance's cost, the cost for hardware repairs (mostly due to hard disks failures) for a datacenter with more than 100,000 servers was estimated to be over a million dollar by a Microsoft's paper published in 2010 (Vishwanath and Nagappan, 2010).

Selected redundancy schemes. We study four different settings for Reed-Solomon codes and three different settings for alpha entanglement codes, see Table 4.6. Replication is not included in this study since the cost of storage overhead to obtain comparable fault tolerance values is too high.

Simulation model. The framework is designed to study what happens with data in environments that either accumulate plenty of failures before repairs take place or where a large number of failures happen all at once. The design is based in four principles: a) easy to adapt for classical (k, m)-codes and alpha entanglements codes, b) should allow us to reproduce results, c) should help us to find answers to questions that were not considered during the development of the simulation and d) should be easy to query and reply fast. Our choice was to generate records that represent all the blocks stored in the system and store them in a database to run diverse SQL queries easily.

We consider these four major aspects:

Left Id	Right Id	Block Type	Location	Available	Repaired
26	26	d	41	True	False
26	31	h	7	True	False

Table 4.8 – Table representation for code AE(3,2,5): d-block d_{26} and p-block $p_{26,31}$ at H strand.

data Tables 4.7 and 4.8 show a simplified version of the tables that contain X-encoded blocks, where X is any of the codes listed in Table 6.1. Each table has two or three columns to identify a block. For example, RS codes use "Stripe" and "Block" and alpha entanglements use "Left Id", "Right Id" and "Left Id" and "Block Type". Column "Location" contains a number that determines the location of the block. Column "Available" indicates the availability of the block. Its value is updated after failures and repairs. Finally, column "Repaired" indicates if the block was repaired. All experiments are done with 1 million data blocks, which means that the number of stripes is different for each code setting, e.g., RS(8,2) creates 125,000 stripes and RS(5,5) creates 200,000 stripes.

placements A simple but realistic way of mapping data blocks to storage devices is random placement. This paper presents results for a medium size system with 100 distinct locations. Therefore, each block is assigned a random number from 0 to 99. The model can be adapted to support other placement policies like round-robin strategy or copyset model, which are discussed in Section 7.3.4.

failures A disaster is simulated by changing the availability of a certain percentange of locations. These figures are given in percentages, e.g. 10-50%. One unavailable location impacts on the availability of many blocks. The column "Available" is updated with this new information.

repairs For each unavailable block a process updates the available column if there are enough blocks to repair it. Some codes require recursively repairs until all blocks are available or irrecoverable pattern failures are found. The columns "Available" and "Repaired" are updated accordingly.

Limitations. Spreading blocks evenly over nodes is a well-known scalability problem that affects load balance and system reliability. When locations are selected at random, not all stripes have their blocks "stored" in different locations. For example, in the case of RS(10,4), the original 1 million data blocks resulted in 0.4 million additional encoded blocks. The 1.4 million blocks were distributed to 100 locations with a mean of 14,000 blocks per site and a standard deviation $\sigma = 130.88$. In a total of 100,000 stripes, only 38,429 had their 14 blocks distributed to different locations. The rest of the stripes are distributed in locations (stripes): 8 (5), 9 (39), 10 (475), 11 (3,746), 12 (17,076), 13 (40,230). As expected, with more locations the blocks of the same stripes are distributed better in different locations, e.g., using 1,000 locations 91,167 stripes have their 14 blocks in different locations. While, at first sight, the distribution of Reed-Solomon coded blocks is not even for a medium size system (100

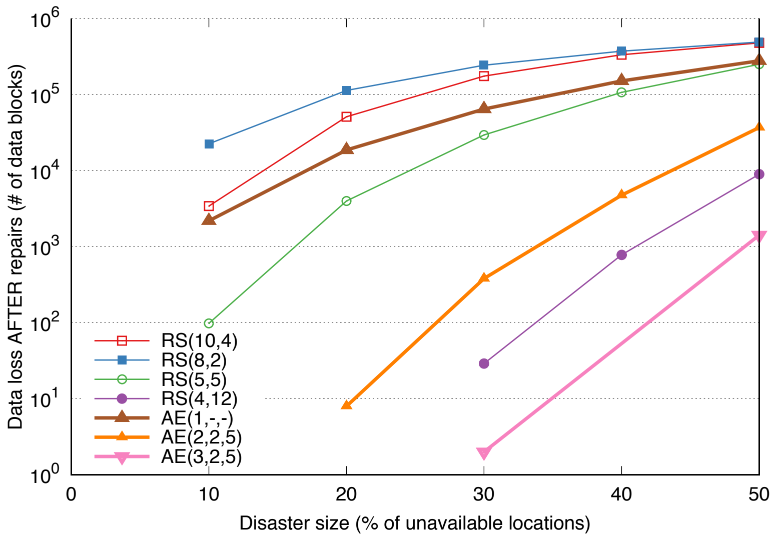


Figure 4.12 – Metric: Data loss. Data blocks that the decoder failed to repair.

locations), we have run other simulations with larger number of locations (1,000) and the comparisons remain close to the ones presented here. An important remark is that the random distribution of α-entanglement encoded blocks in a medium size system may affect the code performance in a worse way since more blocks are interdependent

Metric: Data loss. A stripe is damaged when more than m blocks are destroyed. The data loss metric only counts the data blocks that are at unavailable locations, other available data blocks that belong to damaged stripes are not counted as lost. Figure 4.12 shows that $AE(3,2,5)$ outperforms RS(4,12) even though both are using the same storage overhead. Data loss for AE(1,-,-) is one order more than RS(5,5), a code that uses the same storage overhead, but the gap between both curves decreases when the number of unavailable locations increases. Double entanglements excel at repairing blocks and their storage requirement is equivalent to using triplication. As expected, the curves show that data loss is lower as more storage is used to generate redundancy. Data loss is high when using RS(10,4), RS(8,2), RS(5,5) and AE(1,-,-), in all these settings the additional storage overhead is limited to 100%. These four settings may yield inadequate protection and low availability in unreliable environments unless other measures are taken.

Metric: Vulnerable data. We define the metric of vulnerable data to be the data blocks that are not protected by any other redundant block after the repair process recovers loss data. It illustrates how the level of redundancy in the system decays when only minimal maintenance operations are done. Minimal maintenance happens when the decoder repairs unavailable data blocks but not unavailable parities. However, some unavailable parities are repaired if they are collocated in the same stripe with an unavailable data block. A storage system can receive little maintenance under the following common scenarios: data block repairs are given priority, there is a lack of incentive to recover parities, missing parities may be difficult

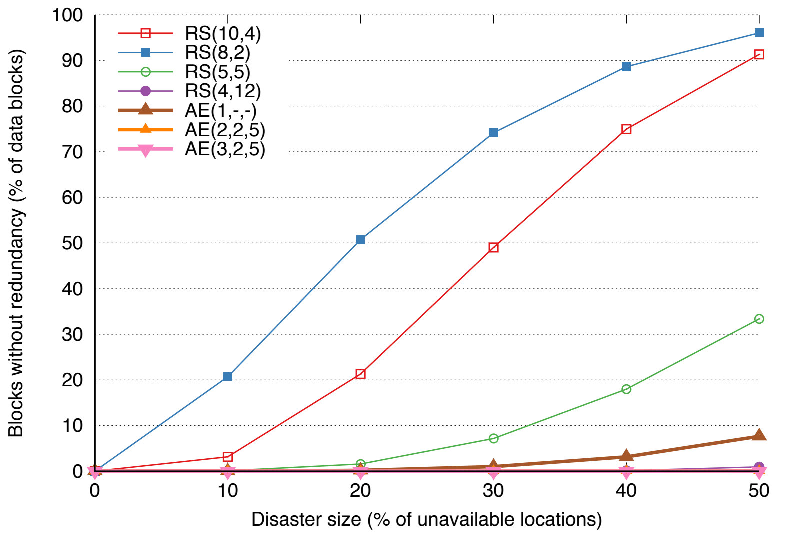


Figure 4.13 – Metric: Vulnerable data. Data blocks without redundancy.

to detect when repairs are triggered by data block read failures. Minimal maintenance can pose a threat to a big portion of the data in a storage system. Figure 4.13 shows the high percentage of data blocks that remain without redundancy when blocks are decoded with Reed-Solomon codes. We see that RS(5,5) performs worse than AE(1,-,-) and leaves more data without redundancy when failures affect more than 20% of the storage locations. This result explains why in Figure 4.12 the gap between the data loss curves for both codes becomes closer in large disaster scenarios. The RS(4,12) code is the only one comparable to the high protection provided by AE(3,2,5).

Our results are somehow related to the impact of combinatorial effects and peer availability effects mentioned by previous authors (Lin et al., 2004). These two effects explain the factors that benefit erasure codes and replication codes in p2p systems. Erasure codes take benefit of the combinatorial effect, however, they are more dependent of the availability of multiple locations. The authors also mentioned that when the peer availability is low, replication can be better than erasure codes. Alpha entanglements are less affected by the peer availability effect since only two blocks are used for repairs but they benefit from the combinatorial effect too.

Metric: Single failures cost. Single failures repairs are very expensive for erasure coded data. When one or more locations are unavailable, stripes are affected unevenly, however, the majority of them will have one single data block loss. In MDS codes, repairing a single-failure is expensive as it requires k times bandwidth overhead and I/O operations at k locations. The concept of single failures in alpha entanglements is orthogonal to the way single failures are seen in RS codes. Single failures are cheap for entangled data since a single data block is repaired with a fixed "k=2" (two parity blocks) for any code setting. Repairs in alpha entanglements are performed in several rounds. Figure 4.14 shows the percentage of data blocks that

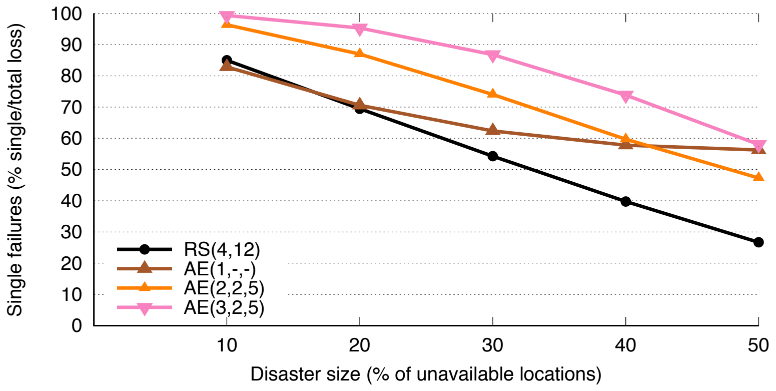


Figure 4.14 – Metric: Single failures. What part of repairs are single-failure repairs?

Code	10%	20%	30%	40%	50%
AE(1,-,-)	6	7	9	10	10
AE(2,2,5)	3	6	9	17	30
AE(3,2,5)	3	4	7	10	15

Table 4.9 – Metric: Code performance in AE codes: Number of repair rounds to recover all repairable data blocks with 10% to 50% unavailable locations.

required single-failure repairs. For alpha entanglements, we computed the ratio between all single failures solved at the first round and all the data loss repaired in the system. The figure shows that a large portion of the failures represent single-failures. Alpha entanglements allow efficient recovering from single-failures by using parallel local repairs. For reference, we compute the same proportion (without considering rounds) for RS(4,12), which is the code with highest locality among our selected RS code settings and superior than other locally repairable code constructions like the HDFS-Xorbas implementation (Sathiamoorthy et al., 2013). For RS codes, the repair efficiency improves when the percentage of unavailable locations increases causing the number of single failures to drop.

Metric: Code performance. This metric is only specific to alpha entanglement codes. At each round, our AE decoder computes 1 XOR between two available blocks for any data and parity blocks that is repaired. When data blocks cannot be repaired at the first round, the decode will do it at the second round if other required data or parity block becomes available. Figure 4.14 shows that most of the data is repaired at the first round. The amount of blocks that is repaired per round decreases abruptly, normally the last round repairs only 1-2 blocks. But, how many rounds are needed to recover all repairable data blocks? This question is addressed in Table 4.9. This table shows that increasing α reduces the number of rounds needed to recover the lattice. For large failure scenarios (40-50%) the improvement is significant. The number of rounds required by the code setting AE(1,-,-) remains constant after reaching a failure scenario in which 40% of the locations are unavailable. The reason for this behavior is because the code is not able to recover more blocks even with more repair rounds.

4.7 Summary

We have presented the modelling, simulation and assessment of complex entanglements. Alpha entanglement codes are a practical and flexible erasure code solution designed for increasing the reliability, performance and integrity of a storage system using redundancy propagation. We have shown that entangled data can survive catastrophic failures, the redundancy level does not degrade as fast as in Reed-Solomon coded data, single failures repair is optimal and the number of rounds required to recover missing data is low. The encoder and decoder are lightweight and efficient, essentially based on exclusive-or operations, and offer promising trade-offs between security, resource usage and performance. In the following chapters, we analyze in more detail the reliability and fault tolerance of the codes, as well as, the features that make entanglements an attractive solution for storage systems.

In the last two decades, many peer-to-peer systems have risen and fallen in popularity. The current research trend on erasure coding is reducing at maximum the storage overhead while improving the efficiency of repairs. However, the main interest of the industry is on reducing storage costs. If we consider the potential savings on maintenance cost, alpha entanglements make sense in a wide variety of scenarios.

5 Fault Tolerance and Reliability

This chapter presents additional studies to understand the fault tolerance and reliability of systems built with complex entanglement codes. We show the impact that code settings variations have on fault-tolerance. Since the code parameters regulate the shape of the entanglement lattice, they specify the way redundant data is propagated. This study presents the outstanding effects of emergent properties of redundancy propagation. Even though the lattice is resilient to a large set of erasure patterns, some data loss is possible when certain patterns are present. Furthermore, the code is irregular, meaning that it can tolerate some patterns of a certain size but not all of them. We analyze erasure patterns and present upper and lower bounds for the data propagation method. To further complete the assessment we present a reliability study based on combinatorial methods. The first part of this chapter delves deeper into the study of alpha entanglements and the second part extends the work published in CLUSTER'15 (Estrada-Galinanes and Felber, 2015b).

5.1 Introduction

As explained by Shannon (Shannon, 1958), communication theory deals with reliability challenges analogous to the ones that appear when building reliable computers. In information theory, one wishes a reliable message transmission even if the reliability of transmitting a single symbol is poor. When building a large scale computing machine, a single error in the components, wiring or programming may cause incorrect behavior at the output. To introduce the circuit networks built with entanglement codes, we choose the opportunity to recognize von Neumman, who made tremendous advances in the design of reliable computing machines by contributing to the field of automata theory. As mentioned by von Neumman (Von Neumann, 1956), Turing (Turing, 1937) and later Mc Culloch and Pitts (McCulloch and Pitts, 1943) proposed that automata are best suited to study intuitionistic logic, where the logical prepositions can be electronic networks or idealized neural networks. Mc Culloch and Pitts' thesis is that in a nervous system "for every net behaving under one assumption, there exists another net which behaves under the other and gives the same result, although perhaps not in the same time". Von Neumann work was influenced by the biological model of computation in the human brain. He also designed self-reproduction systems in order to accelerate the process of reprogramming the EDVAC (Von Neumann, 1993) machine. One of his models, which was described in an unfinished work, may suggest the biological reproduction at the cellular or molecular level. His model is represented by an infinite array of cells, and the cells can be in twenty-nine possible states. The state of the cell at a given time is a function of its value at the preceding time and the value of its four neighbor cells at the preceding time. The state of the cell changes as time progresses, and with proper construction it is possible to make a certain group of neighbor cells to act as a living entity that can keep its identity (a pattern of "active" cells) moving and causing that other cells take a similar pattern.

The above mentioned work brought enormous contributions in many areas. In particular for the design of a highly reliable system, it means that an engineer could combine components in serial and parallel circuits in such a way that the whole system is more reliable than its component parts. This bring us to the circuits design in complex entanglement codes. Complex entanglement codes build a complex network of interdependent elements that propagates redundant information to neighbor elements as time progresses and new elements are “ingested” by the system. The circuits and cycles defined in a lattice permit reading an element following a certain path or another path, perhaps longer, therefore requiring more computation. The possible paths are determined by the entanglement code parameters. The paths themselves do not determine completely the system’s reliability since at the application level the engineer has the freedom to design different map algorithms to allocate the elements in storage devices that build a reliable storage system.

In this chapter, we seek to complete the assessment of complex entanglements with an analysis of failure patterns. The motivation is to understand how the code parameters impact on the fault-tolerance of the code. Finally, we complete our evaluations with a reliability study based on combinatorial methods. We undertook this study to compare helical entanglement codes with other redundancy methods. Classical redundancy methods such replication and Reed-Solomon codes can be modeled as parallel and “k out of n” redundant system respectively. Entanglement codes are not an exception. They can be modeled as a combination of serial and parallel paths. Based on this serial/parallel configurations, system reliability can be computed and expressed as a failure rate (Faraci, 2006).

5.2 Code Settings and Fault Tolerance

Codes that provide regular fault tolerance tolerate all erasure patterns that are smaller than the Hamming distance d. On the opposite, codes that provide irregular fault tolerance tolerate more than the "m" failures tolerated in (k,m)-codes but the extra tolerated failures are not arbitrary. Irregular codes are usually compared according the distribution of these patterns. In information theory, the weight distribution of a code is given by computing the number of codewords of Hamming weight i for $d \leq i \leq n$ with n the length of the code. Several researchers came up with a variety of other metrics while studying irregular codes in the context of storage applications. A number of studies used the minimal erasure list (MEL) and the fault tolerance vector (FTV). A minimal erasure is an irreducible pattern that causes the loss of data and parity blocks. The MEL (Wylie and Swaminathan, 2007) is the enumeration of all minimal erasures in irregular XOR-based flat codes. In that work, Wylie proposed the MEL algorithm. The FTV (Greenan et al., 2008a) aggregates the information from the minimal erasure list and computes the probability that data is lost given a number of failures. FTV is the complement of the conditional probability vector (Hafner and Rao, 2006), which expresses the present state at a state k in a reliability Markov model.

α-entanglements are *irregular codes*. We use a variation of Wiley's study to gain more knowledge about the pattern size and its impact on data block loss. Wiley focused on characterizing failure patterns size without making the distinction between pattern size and actual data loss. Our goal is to characterize the size of the minimal erasure patterns that cause a certain amount of data loss. Our modification includes the notion that an erasure pattern of size y blocks only has x data blocks, with $y > x$. *Ideally, we want patterns with $y \gg x$ since that means a high fault-tolerance, which decreases the probability that the decoder fails, but when it fails only a small fraction of blocks are data blocks.* For many patterns we can increase the ratio y/x using the entanglement code parameters. An outstanding characteristic is that the entanglement parameters s and p permit increasing the ratio y/x without generating more storage overhead or increasing the repair cost of single failures. The increase in fault tolerance when tuning the code parameters is evidenced by a cross-study comparison of the size of minimal erasure patterns for multiple code settings. This study does not identify all erasure patterns for α-entanglements. To minimize the burden of such task, we concentrate only on the most relevant patterns to determine which of them have lower and upper bounds for the redundancy propagation achieved by entanglement codes. The term lower bound means the smallest possible size for an arbitrary minimal erasure pattern that impacts on x data blocks, i.e., fault tolerance for x cannot become worse (since changing the code parameters cannot reduce the size for patterns affecting exactly x d-blocks). The existence of upper bounds implies that there are patterns that impact on x d-blocks that maintain their size when increasing s and/or p, i.e., the lattice structure at a local level only changes with α.

The motivation is to understand how the code parameters impact on the fault-tolerance of the code. If the erasure pattern becomes larger but the number of actual erased d-blocks remains constant, it shows evidence of the code's fault-tolerance improvement.

5.2.1 Problem Complexity

We could not find in the literature a technique that help us to identify erasure patterns with the level of detail described above. Our results are obtained by visual inspections on the lattice and verifications can be done with a tool that was implemented in Prolog language (see Appendix A). The problem is tractable if we evaluate the size of ME for small numbers of x d-blocks. We limit our study to $x \in [2,8]$ in many different code settings . These are the smallest and yet the most relevant patterns for entanglements with $\alpha \leq 3$.

The influence of the smallest patterns in fault tolerance is easily observed in the FTV metric, whose i-th entry is calculated as by

$$\mathrm{FTV}_i = \frac{e_i}{\binom{n}{i}}\,, \tag{5.1}$$

where e_i is the total number of patterns of size i that cause data loss and the denominator is the total possible combinations of patterns of size i in a code of length n. In particular, if we assume that e_i is small, it is safe to focus only on the small patterns because the denominator becomes very large when the number of d-blocks tends to infinity. At a threshold $i = j$ the numerator and denominator will become comparable again but those cases are not interesting for the task of identifying the lower and upper bounds mentioned before. Discarding the larger patterns simplifies our analysis while leaving space to deepen the study on how a particular pattern evolves with different code settings. To reduce more the problem complexity, we adopt a divide and conquer approach. It is easier to start with the analysis of the most basic patterns that cause data loss in single entanglements, namely, *primitive forms*. Then, we combine them to construct more complex patterns that serve to understand pattern behaviors.

Concerning the selection of the graph model, the Tanner graph (Tanner, 1981) is the most predominant graph representation among studies for constructing long codes from one or more shorter codes. The Tanner graph is a bipartite graph composed by variable nodes and check nodes. It has a compact design but it does not distinguish between our p-blocks and d-blocks. Our first attempt to provide the Tanner graph representation for entanglement codes was abandoned since both blocks would be represented identically as variable nodes. This abstraction does not permit the detailed study that we want to conduct. Consequently, this pitfall prevent us from using Wylie's MEL algorithm since the program takes as input the code's Tanner graph. Instead, we use a lattice graph like the representation given in Figure 4.6 and identify the induced subgraphs that are erasure patterns. We find that this is the most convenient way of showing the impact of code parameters on the size of erasure patterns.

To conclude, Wylie and Greenan conducted quantitative analysis on the fault tolerance of many XOR-based codes (Wylie and Swaminathan, 2007; Greenan et al., 2008a). That gives a complete idea of the size and quantity of the failure pattern for a particular code. We opt for a qualitative analysis on a small set of erasure patterns. That leads to holes in our knowledge but sheds light on an important aspect of the code that would be hidden in a quantitative study like other researchers did. Having said that, both approaches are valid and complementary.

5.2.2 Basic Notation and Preliminaries

We say that the lattice tolerates a certain erasure pattern if our decoder successfully recovers all the blocks that form the pattern. In this section we study the erasure patterns which cannot be repaired. The terms erasure pattern and minimal erasure (ME) are a variation of Wylie (Wylie and Swaminathan, 2007). MEs are irreducible patterns in the sense that the removal of any of their blocks [1] opens the possibility to recover the erased blocks. In the context of alpha entanglements, an erasure pattern $ME(x)$ is a set of erased blocks that results in $x \geq 2$ irrecoverable d-blocks. Erasure patterns that result in a single irrecoverable d-block are disregarded. Erasure patterns composed by a single d-block can only occur at the beginning or end of an open entanglement chain or lattice. However, those can be protected with additional replication, hence, they are discarded for this study.

We introduce the notation $|ME(x)|_A = y$ to say that the size of the *minimal erasure* that causes the loss of x d-blocks using code A is y. Data loss is due to irreversible damage in the lattice structure defined by a code A with parameters α, s and p. **If the size of the pattern for a certain x becomes bigger by changing s and p it means that we are increasing fault tolerance without increasing the storage or repair costs.** Considering Equation 5.1, if now $ME(x) = y'$ the pattern will be counted at FTV_j instead of FTV_i, where $i = y$, $j = y'$ and $y' > y$. In other words, a larger pattern means that the failure needs to impact on more elements in the system to cause the same data loss. The storage overhead and repair costs remain constant since α has the same value, i.e., the number of parities in the system is the same and single failures are repaired with the same amount of blocks.

Our study shows that in many cases there is a positive correlation between x and y, but unveils erasure patterns that have a critical behavior with some code parameters. The study is limited to the most relevant erasure patterns, i.e., the ones that cause a small amount of data loss, hence, with a small $|ME(x)|$.

Finally, we distinguish two type of nodes that describe the state of nodes (d-blocks) members of an erasure pattern:

dead nodes are irrecoverable nodes members of the pattern.

live nodes are available nodes or otherwise repairable nodes which are not actually counted in $|ME(x)|$ but serve to define the graph that describes the pattern.

In single entanglements, if a live node is erased, it becomes a dead node. But in lattices, if a live node is erase, it may be repairable through any of the possible α directions that can be taken at any data node.

5.2.3 Primitive Forms

Primitive forms are the simplest erasure patterns. The following definitions will provide the basic understanding on the conditions that need to be met for experiencing data loss:

Definition 5.2.1 (Permanently erased node - dead node)**.** Let $G = (V, E)$ be a lattice built with

[1]When we do not specify, the term block refers indistinctly to d-blocks and p-blocks.

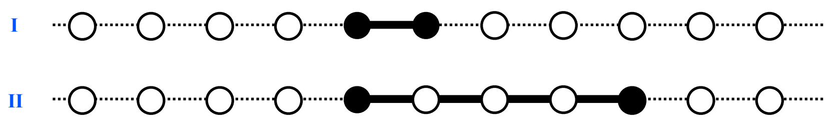


Figure 5.1 – Primitive forms cause the loss of two data blocks in single entanglements. Black filled nodes and thick edges indicate unavailable elements that describe a failure pattern. The shortest form I is built only with dead members. The larger form II includes inner live members.

α-entanglements. A node $v_i \in V$ is permanently erased, namely a *dead node*, *iff* v_i and α incident edges e_{ij} are permanently erased.

Note that the edges e_{ij} belong to α disjoint edge sets.

Definition 5.2.2 (Permanently erased edge)**.** Let $G = (V, E)$ be a lattice built with α-entanglements, let $E = E_1 \cup E_2 \cup \cdots E_\alpha$ and let $E' \subset E_k$, whose edges describe a path of adjacent edges that connect two endpoint dead nodes. An edge $e_{ij} \in E'$ is permanently erased, *iff* $\forall e_{ij} \in E'$, e_{ij} is erased.

Now, let's define a primitive form as follows,

Definition 5.2.3 (Primitive form)**.** Let $G = (V, E)$ be a lattice built with α-entanglements, let $E = E_1 \cup E_2 \cup \cdots E_\alpha$, let $E' \subset E_k$ and let $S_P \subset V$ a subset of adjacent nodes of at least two nodes in a path described by E'. A connected induced subgraph $G[S_P] = (S_P, E')$ is a primitive form *iff* at least the two endpoint nodes are dead and all the edges $e_{ij} \in E'$ are permanently erased in the lattice G.

Figure 5.1 shows two primitive forms. *Form I* has size 3 because it involves the loss of two data nodes (d-blocks) sharing an incident edge (p-blocks), whereas *Form II* has size 6 as it is composed by two dead nodes and four edges defining the path that connects them.

Primitive forms can cause data loss only in single entanglements. To elaborate more on this topic we define complex forms and minimal erasure patterns.

5.2.4 Complex Forms

A *complex form* is a connected graph built with multiple primitive forms. It is useful to represent erasure patterns, with the characteristic that the live nodes are not counted in the size of the pattern.

Definition 5.2.4 (Complex form)**.** Let the lattice be a graph $G = (V, E)$ constructed by α-entanglements with $\alpha \geq 2$, let S_{P_i} be a primitive form, let $S_C \subset V$ be $S_C = S_{P_1} \cup S_{P_2} \cup \cdots S_{P_j}$, and let $G[S_C] = (S_C, E')$ the connected induced subgraph of G. A subgraph of $G[S_C]$ $H = (S_C, E'')$ is

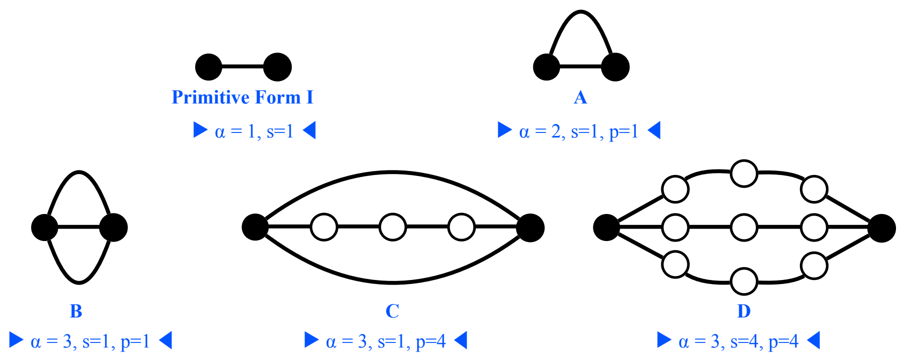


Figure 5.2 – Four examples of complex forms. The figure shows the changes to the primitive form $ME(2)$ with different code parameters.

a complex form *iff* the edge set E'' contains the edges incident to a node pair belonging to S_C, namely all the edges e_{ij} incident to dead nodes and the edges e_{kl} incident to at least one live node if $(v_k, v_l) \in S_{P_i}$.

Figure 5.2 illustrates the definition of complex forms by presenting four examples from distinct lattice configurations. Complex form A is a subgraph of a double entanglements lattice AE(2,1,1). Complex forms B-D are subgraphs of triple entanglement lattices with different s and p parameters: AE(3,1,1), AE(3,1,4) and AE(3,4,4), respectively. We can make two observations:

Multiple paths increase fault tolerance. First, increasing α increases the number of paths from one vertex to another, hence, increases the degree of dead nodes. The cardinality of the edges set of a given form, denoted |E(H)|, increases as α increases: $|E(I)|_{\alpha=1} = 1 < |E(B)|_{\alpha=2} = 2 < |E(C)|_{\alpha=3} = 3$.

Longer paths increase fault tolerance. Second, increasing s and p increases the length of a path connecting two dead nodes. The number of edges in the horizontal path increases as p increases, efficiently increasing $|E(H)|$: $|E(B)|_{\alpha=3,s=1,p=1} = 3 < |E(C)|_{\alpha=3,s=1,p=4} = 6$, and the number of edges in the helical paths increases as s increases: $|E(B)|_{\alpha=3,s=1,p=4} = 6 < |E(D)|_{\alpha=3,s=4,p=4} = 12$. E(H) grows efficiently because the number of p-blocks needed to increase the code parameters s and p remains constant.

5.2.5 Minimal Erasure Patterns

A minimal erasure pattern $ME(x)_A$ is an irreducible pattern for code setting A that causes the loss of x d-blocks in A. It is represented with a complex form H=(V,E) but if V includes live nodes, $|ME(x)|$ does not include them. Live nodes are not part of the size of the minimal erasure pattern since their availability does not impact in the availability of the other elements in the graph. Because live nodes are part of other strands, which are not part of the pattern, they could be easily repaired in case of being erased. On the contrary, the decoder cannot recover the dead nodes because all the edges that connect them are destroyed. In this way, the size of a minimal erasure pattern is $|ME(x)| = x + \delta$ where δ are the minimum number of permanently erased edges in a x dead nodes pattern.

At this stage, we can recap what we know about minimal erasure patterns.

Theorem 5.2.1. Let $\forall v_i \in S_P$, v_i be a dead node. A primitive form $G[S_P]$ is a minimal erasure pattern ME(x) *iff* $\alpha = 1$.

Proof. Seeking a contradiction, suppose that $G = (V, E)$ is a lattice constructed with $\alpha > 1$, and suppose that the primitive form $G[S_P] = (S_P, E')$ is a minimal erasure pattern. According to Definition 5.2.1 a dead node v_i has its α incident edges permanently erased. But node v_i has $\alpha - 1$ edges $e_{ij} \notin E'$, this contradicts the supposition that $G[S_P]$ is a minimal erasure pattern. □

Theorem 5.2.1 tells us that *Form I* (Figure 5.1) is a minimal erasure pattern. In fact, Form I is the sole irrecoverable triple failure with $|ME(2)| = 3$ presented in the vulnerability analysis done for single entanglements in Chapter 3. Form II is *irreducible* in the sense that all erased elements must be part of the pattern, and if one is removed all other elements can be recovered. However, its size $|ME(2) = 6|$ is larger than the former since the graph contains live nodes that augment the number of edges in the pattern.

All the complex forms shown in Figure 5.2 are minimal erasure patterns for the indicated lattice configurations. Now, it is possible to complete our previous observations:
Multiple paths increase fault tolerance. $|ME(2)|_{\alpha=1} = 3 < |ME(2)|_{\alpha=2} = 4 < |ME(2)|_{\alpha=3} = 5$. As a consequence, an arbitrary pattern of three elements may cause data loss in a single entanglement but not in lattices with $\alpha > 1$. Note that the compared values with $\alpha > 1$ have $s = 1$ and $p = 1$.
Longer paths increase fault tolerance. The improvement is shown in the examples of Figure 5.2 $|ME(2)|_{\alpha=3,s=1,p=1} = 5 < |ME(2)|_{\alpha=3,s=1,p=4} = 8 < |ME(2)|_{\alpha=3,s=4,p=4} = 14$.

5.2.6 Lower and Upper Bounds

In summary, we know that a larger α degree results in more paths, and larger s and/or p parameters result in longer paths. *But, are there any limits?* We search for lower and upper bounds in the size of $ME(x)$ to completely characterize erasure patterns. Our study was verified on 19 different lattices with $x \in [3, 8]$.

In an entanglement lattice ($\alpha > 1$), the lower theoretical bound for $|ME(x)|$ is nothing else than the number of dead nodes x plus the number of edges in a α-regular graph defined with x vertices.

Theorem 5.2.2 (Lower bounds for propagation)**.**

$$|ME(x)| \geq x + |E[\alpha\text{-regular graph}]|.$$

Proof. In a lattice built with α-entanglements, a node v_i is permanently erased if v_i and its α incident edges are permanently erased. An edge is permanently erased if $\forall e_{ij} \in E(G[S_{P_k}])$ are permanently erased. To erase multiple x nodes with the minimum number of elements, all the primitive forms $G[S_{P_i}]$ must be the shortest possible (ideally with lengh 1) the nodes must describe the smallest possible complex form to represent a minimal erasure pattern $ME(x)$. The smallest theoretical complex form is an α-regular graph with x vertices (nodes). □

The number of edges for an α-regular graph can be computed from the Euler's theorem (sometimes known as *the handshaking theorem*) that we transcribed here:

Theorem (Degree sum formula)**.**

$$\sum_{v \in V} \deg_G(v) = 2|E|,$$

where $\deg_G(v)$ is the degree of a vertex v. This theorem says that the sum of all vertex degrees is equal to twice the number of edges, since each edge contributes exactly 2 to the sum. In particular for r-regular graphs, if r is odd then the number of edges must be divisible by r.

We have to be careful when computing $|ME(x)|$ when α and x are simultaneously odd. Euler's theorem tell us that if the sum of the degrees of vertices with odd degree α is even, there must be an even number of those vertices. In other words, a graph with odd degree and odd number of vertices does not exist. For our purpose, we found examples of complex forms $ME(x)$ for α and x being odd that have the lower bound in Theorem 5.2.2. Those graphs are not an α-regular graph. They have one or more nodes that have degree $r \in [\alpha, 2\alpha]$. To compute the theoretical lower bound when the node degree α is odd and the number of erased nodes is odd, the number of edges must be a multiple of α.

Corollary 5.2.1 (From Theorem 5.2.2)**.** The lower bound for $|ME(x)|$ when α and x are odd is computed with $|ME(x)| \geq x + |E[\alpha\text{-regular graph}]|$.

Proof. From Euler's degree sum theorem, we know that it is not possible to find a r-regular graph $G = [V, E]$ with odd r and odd $|V|$. But, if the graph has one or more outlier nodes v with degree $\deg(v) > r$ adjacent to nodes u with degree $\deg(u) = r$, the number of edges is equivalent to the edges computed with Euler's theorem for an ideal r-regular graph since edges must be counted only once. □

Figure 5.3 shows minimal erasure patterns $ME(x)$ with $x \in [3, 8]$ for triple entanglement lattices. These examples have the characteristic that they have the smallest number of edges, hence, they meet the lower bound in Theorem 5.2.2. It is not possible to find another subgraph for the same lattice configuration with a smaller size. But, the examples may not be unique as

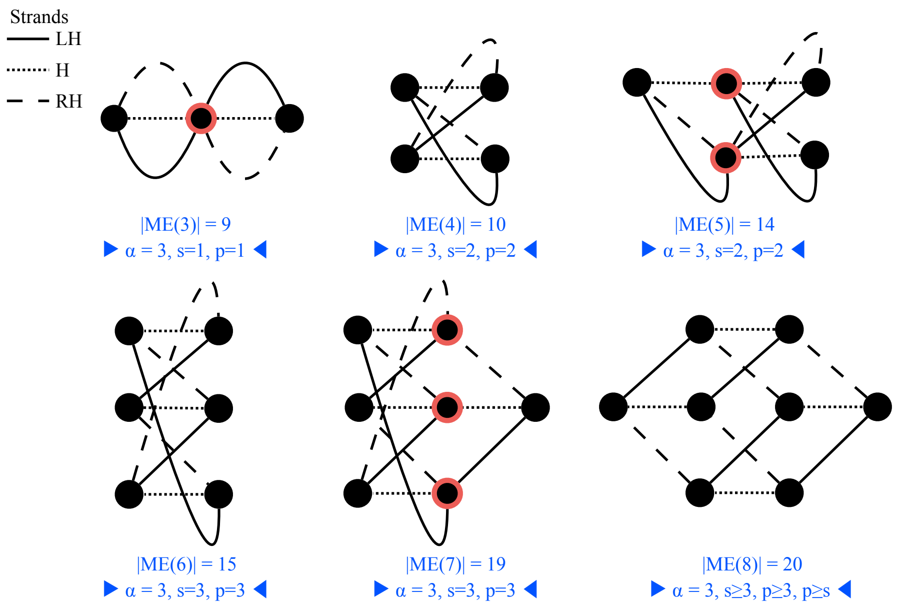


Figure 5.3 – Examples for $ME(x)$ with $x \in [3,8]$ having the theoretical lower bound for $|ME(x)|$. If x is odd, $ME(x)$ graphs have outlined vertices that lost more than α incident edges. If one of the "extra" edges is removed from the subgraph, its neighbor vertex is recoverable, and recursively all the rest.

isomorphic subgraphs, which have identical $|ME(x)|$, may exist. In all the examples for odd α, the $|ME(x)|$ for odd x was computed according to Corollary 5.2.1 and with the corrections indicated above. For example, $|ME(5)| = 5 + |E[\text{3-regular graph}]|$ uses a 3-regular graph with 5 nodes although it does not exist. Using Euler's theorem for an ideal 3-regular graph $|E| = 7.5$ or $|E| = 9$ when it is corrected to be multiple of α, as a result $|ME(5)| = 14$. The real graph has a node v_1 with $\deg(v_1) = 4$ and a node v_2 with $\deg(v_2) = 5$. Various cases for odd $|V|$ are shown in the figure with outlined vertices to indicate the exceptions.

Our formula for lower bounds is in agreement with the observations given for Figure 5.2. For triple entanglements, the complex form B is the graph that meets the lower bound for $|ME(2)|$. When the lattice changes with larger s and/or p, $|ME(2)|$ increases, thus, fault tolerance improves.

A final observation regarding lower bounds is that the code's fault tolerance is more impacted by the smallest $|ME(x)|$ for any arbitrary x. For certain configurations, there exists $x > y$ such that $|ME(x)| < |ME(y)|$. Figure 5.4 shows an example for $|ME(4)| < |ME(2)|$ in the code AE(3,2,9).

The next topic to address is the upper limit for $|ME(x)|$. To put it in other terms, we want to determine the $ME(x)$ whose $|ME(x)| = c$ remains invariant with respect to lattice configuration changes. To understand upper limits we need to study the limitations embedded in the lattice geometry.

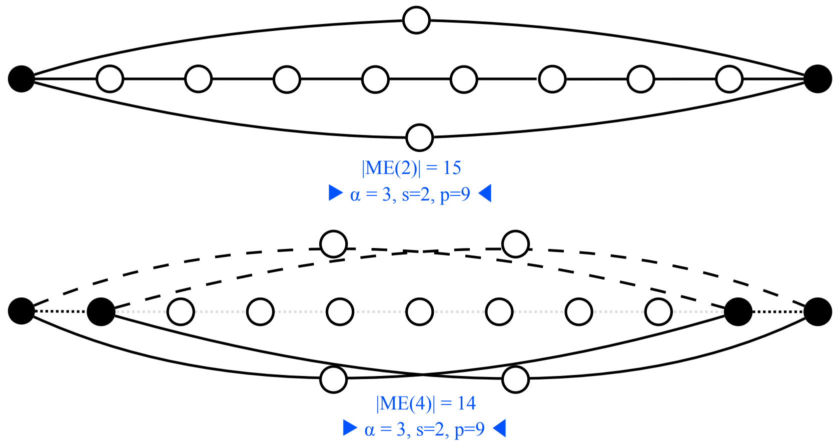


Figure 5.4 – Critical erasure patterns: $|ME(4)|$ is smaller than $|ME(2)|$ for the same code settings.

Lattice Geometry Upper Bounds

The geometry of the lattice that models the redundancy propagation is defined by the code parameters (α, s, p). In very rough terms, s defines the number of rows, p defines the number of distinct columns and α defines the number of directions in which information is propagated. In a description that has close resemblance with computational physics, the geometry can be described using its propagation units: "linear", "square", "cubic", and generally as "α-hypercubic". A lattice of n nodes is composed by n distinct propagation units, with the ones at the right extremity not yet completed. The lattice grows regularly while the incomplete propagation units become closer to completeness and new incomplete ones are added. Figure 5.5 shows examples of square and cubic propagation units.

A formal definition for propagation units is:

Definition 5.2.5 (Propagation unit)**.** Let the lattice be a graph $G = (V, E)$ constructed by α-entanglements, and let $\Omega \subset V$ be a subset of nodes with $|\Omega| = 2^{\alpha}$. The nodes $v \in \Omega$ are a propagation unit *iff* they are the vertices of one of the α-hypercube embedded in the lattice geometry and described by the α-regular induced subgraph $G[\Omega] = (\Omega, E')$ whose edge set $|E'| = \alpha 2^{\alpha-1}$.

Having established the definition for propagation units, we enunciate the following lemma:

Lemma 5.2.1 (Number of primitive patterns in a complex form)**.** Let H=(V,E) be a complex form in a lattice built with α-entanglements. The maximum number of primitive forms that form H is $\alpha 2^{\alpha-1}$.

Proof. A complex form is defined for x dead nodes. We know that the complex form must include α primitive forms for each added dead node. At the lower bound, each primitive

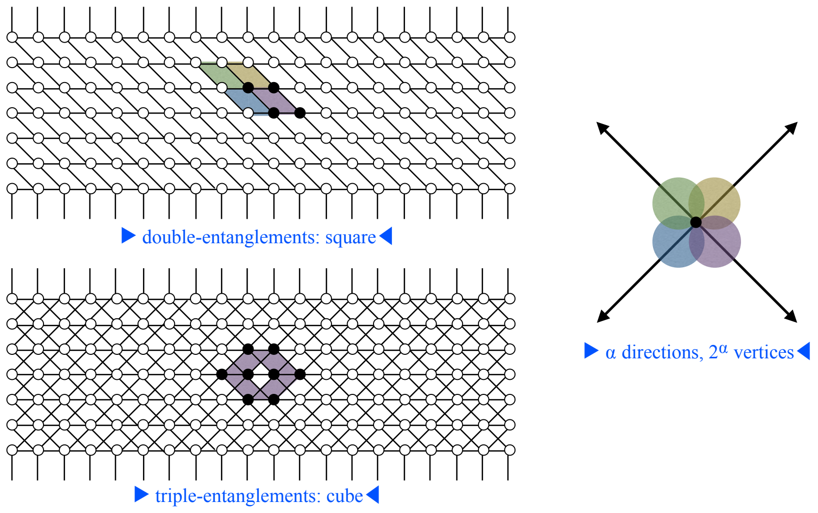

▶ double-entanglements: square ◀
▶ triple-entanglements: cube ◀
▶ α directions, 2^α vertices ◀

Figure 5.5 – Redundancy propagation occurs along α directions through overlapping propagation units of size 2^α.

form is composed by a single edge. The primitive form connects two dead nodes, but the handshaking theorem tells us that the primitive form has to be counted only once. A closed figure that connects x elements in subsets of α elements is an α-hypercube. An α-hypercube contains $\alpha 2^{\alpha-1}$ edges. □

Thus, the upper bound for propagation based on the definition of propagation units is:

Theorem 5.2.3 (Upper bounds for propagation)**.** The induced subgraph $G[\Omega]$ is a minimal erasure pattern $ME(x)$ with $x = 2^\alpha$ and $|ME(x)| = 2^\alpha + \alpha 2^{\alpha-1}$ invariant in s and p unless $s < \alpha$.

Proof. In a lattice built with α-entanglements, a node v is permanently erased if v and α incident edges belonging to α different primitive forms are permanently erased. Using Definition 5.2.5, we know that the set Ω are the vertices of an n-hypercube. The n-hypercube is an α-regular graph that has n vertices v with $\deg(v) = \alpha$, with each of its incident edges belonging to α different paths. If all the elements of $G[\Omega]$ are erased, the decoder cannot recover them. Using Definition 5.2.4 and Lemma 5.2.1, $\forall S_{P_i} \subset E(\Omega)\ |S_{P_i}| = 1$. Thus, the n-hypercube is the smallest theoretical complex form. The connections of a propagation unit are embedded in the lattice geometry and its size does not change with s and p, with the exception of cases where $s < \alpha$ because of changes on the lattice topology. □

Note that the upper limit for propagation is not tight:

Corollary 5.2.2. Each node v in the lattice belongs to 2^α propagation units that share some vertices.

Proof. Theorem 5.2.3 says that the information spread in $G[\Omega]$ is unique. If all the elements of $G[\Omega]$ are erased, the decoder cannot recover them with the partial redundant information outside $G[\Omega]$. On the other side, each node belongs to adjacent propagation units. Because the encoding is done in a regular fashion and a propagation unit has 2^α nodes, we can safely infer that each node can take different positions in 2^α adjacent propagation units. □

An upper limit for propagation is the existence of an erasure pattern that causes data loss even though partial information of the erased blocks is spread in the lattice. This upper limit prohibits the use of s and p to increase the size of this particular erasure pattern, $|ME(2^\alpha)| = \alpha 2^{\alpha-1}$. There are, however, other patterns that are still improved by tuning the code parameters. Most importantly is that the notion of redundancy propagation still applies to cross-boundaries between adjacent propagation units.

All-or-nothing Integrity Bounds

Initial work on entanglements (Aspnes et al., 2007) gave a definition for a storage system with *all-or-nothing integrity* (AONI) to formalize the notion of an AONI storage system, in which either all users can recover the data or no user does. That approach was a first step on the development of entanglement theory, however, it is not well suited for practical applications in storage systems since it hides the complications and costs of providing AONI to all data blocks (or documents). We revised the definition of all-or-nothing integrity to restrict the property to a subset of blocks and adapted the definition to our entanglement codes. In this case, the notation $Pr_E(Q)$ denotes the probability that an event Q occurs when E is the erasure pattern in the lattice caused by a system's failure. The recovery vector $\vec{r}$ summarizes which data blocks can be recovered with the decoding algorithm, with $r_i = 1$ if the data block is recovered and zero otherwise. For example, a subset of blocks d_1, d_2 and d_3 has the recovery vector $r = 110$ if after the decoder finishes its task only d_1 is recovered.

Definition 5.2.6 (AONI)**.** A subset of d-blocks $D = \{d_1, d_2, ...d_n\}$ that belong to a helical lattice encoded with α-entanglement codes is all-or-nothing with respect to a class of erasure pattern $\mathscr{E}$ *iff* for all $E \in \mathscr{E}$,

$$Pr_E(\vec{r} = 0^n \vee \vec{r} = 1^n) = 1$$

With this groundwork in place, we can now state a corollary from Theorem 5.2.3 for the upper bounds of all-or-nothing-integrity:

Corollary 5.2.3 (AONI bounds)**.** Let Ω be an arbitrary propagation unit of a lattice built with α-entanglements. Its α-regular induced subgraph $G[\Omega]$ has the all-or-nothing property.

It asserts that a set of data blocks that define a propagation unit in a lattice for a certain α are protected with all-or-nothing integrity.

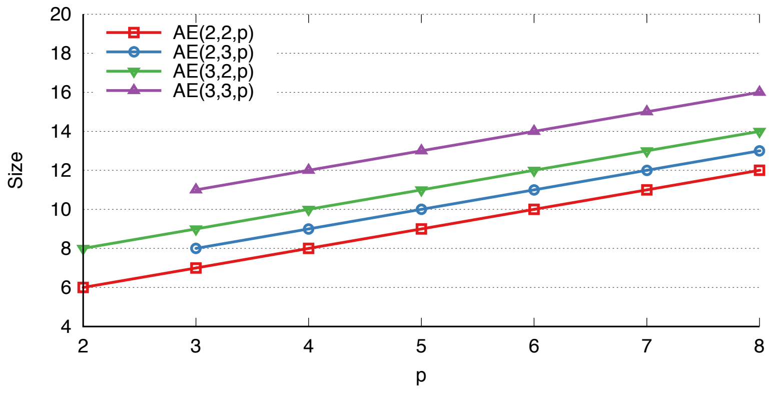


Figure 5.6 – |**ME**(2)| increases with larger s and p.

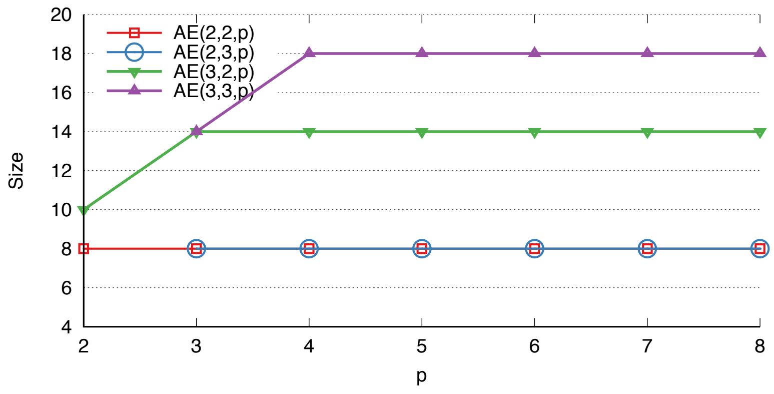


Figure 5.7 – |**ME**(4)| remains constant for $\alpha = 2$, and increases with s for $\alpha = 3$.

Proof. Let $\Omega = \{d_1, d_2, ...d_n\}$ be a propagation unit. Then, Theorem 5.2.3 says that $G[\Omega]$ is a minimal erasure pattern E that belongs to the class $ME(2^\alpha) = \alpha 2^{\alpha-1}$. Let E' a pattern that contains all elements of E except one (arbitrary chosen). Because a minimal erasure pattern is irreducible, $E' \notin ME(2^\alpha) \Rightarrow Pr_{E'}(\vec{r} = 1^n) = 1$. Then, for all $E \in ME(2^\alpha)$ $Pr_E(\vec{r} = 0^n \vee \vec{r} = 1^n) = 1$. Thus, Ω is all-or-nothing with respect to the class of erasure patterns $ME(2^\alpha) = \alpha 2^{\alpha-1}$. □

5.2.7 Results

The curves presented here are examples of how the size of minimal erasure patterns change with different code settings. Figure 5.6 reveals the positive correlation between the size of the minimal erasure pattern $|ME(2)|$ and the code parameters. The study uses step increments in the code parameters $\alpha \in [2,3]$, $s \in [2,3]$ and $p \in [2,8]$. Figure 5.7 illustrates the size changes for $|ME(4)|$. This example shows the upper bound in propagation due to the square and cube embedded in the lattices for $\alpha = 2$ and $\alpha = 3$ respectively. For double entanglements the curves are completely flat but for triple entanglements the size of the pattern is smaller when $s = p$. This is due to the appearance of minimal erasure patterns $ME(4)$ similar to the graph presented in Figure 5.4. The same figure shows that for triple entanglements, $|ME(4)|$

increases with s. In both figures we can observe that $|\mathrm{ME}(x)|$ is minimal when $s = p$. Although the code can provide more fault-tolerance when p increases, the setting $s = p$ cannot be judged as good or bad per se. Trade-offs between performance and reliability should be considered to choose the best parameters.

Appendix A presents more results and an algorithm to verify patterns.

5.3 Reliability Analysis

In this section we assess entanglement codes from a different angle. This time the focus is on the reliability of an entangled system. System reliability can be defined as the probability that the system operates correctly during a period t. We note that correct operation in a storage system imply that the system is able to recover data even in the presence of node and disk failures. The experiments and simulations described in Section 4.6 showed the high data survivability of entangled data. We found that entanglement codes performed better than classical codes in many realistic storage scenarios. This section expands the comparison.

Estimating the reliability of a system means estimating the probability R(t) that the system will operate correctly over the time interval [0, t] given that it operated correctly at time t = 0. If the system is made of two redundant independent elements arranged in parallel, the estimation is straightforward but the computation could become very difficult in a complex network. A widespread mathematical model to estimate reliability is the Markov model. The assumption behind this technique is that the process is memoryless. In other words, the transition to a future state of a system is controlled only by its current state. Moreover, it assumes that failures and repairs are Poisson (exponentially) distributed, i.e., failures and repairs occur at a constant average rate. Another assumption is that the number of events occurring at a certain time interval is independent of the number of events occurring in any other interval. The equations that handle transitions include the probabilities that the system will survive a certain number of failures. Thus, it requires to know all erasure patterns that cause data loss. For more information, we refer to the Electronic reliability design handbook MIL-HDBK-338-B (Handbook, 1982).

The study of fault tolerance for simple entanglement codes is relatively easy. The number of erasure patterns relevant for the study is low and constrained to combinations of missing elements in a single chain. This analysis has been done in previous work (Estrada-Galinanes and Felber, 2015b). Knowing the patterns permits the estimation of the system reliability with Equation 3.5. Chapter 3 presented the estimated reliability for various disk array configurations using a Markov model. The Markov model for simple entanglements uses a six state transition probability diagram that represents the possible states in which the system can be in.

The task of designing a Markov model for complex entanglements is cumbersome. The number of erasure patterns relevant for the study is larger than for simple entanglements. As presented in the previous section, measuring fault-tolerance in detail is difficult. The number

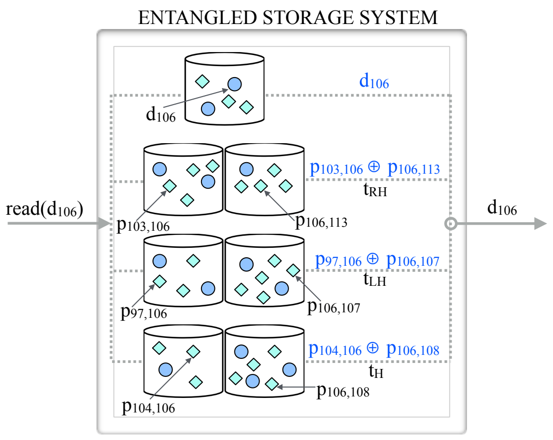


Figure 5.8 – Serial and parallel configurations in an entangled storage system. Each strand connection is a serial configuration composed by the elements in one of pp-tuples (t_{RH}, t_{LH} or t_H). The subsystem is available if both disk nodes are operational. The read request is satisfied if at least one of the four parallel subsystems is available.

of relevant erasure patterns grows exponentially with the number of chains. Besides, complex entanglements are designed to tolerate correlated failures. But, the common assumption for Markov models that failures are independent events and exponentially distributed is not adequate to prove that complex entanglements can tolerate catastrophic failures.

The techniques used in this section are based on graph models of communication networks, as discussed in (Shooman, 2003). We estimate the reliability of a system that responds to one-block (data block) read requests. A system that stores data redundantly can respond with the actual data or using one or more combinations of redundant blocks to decode the requested data. Chapter 4 presented multiple ways of recovering the same content for entanglement codes. Basically, the number of possible recovery trees for the same d-block root grows exponentially with α.

5.3.1 Reliability Concepts and Notation

Our targeted systems consist of a combination of elements in serial and parallel configurations. We begin our analysis by constructing the two fundamental diagrams. Each element or unit represents a disk node that is contacted during the process to read a d-block or a p-block. We use the term *disk node* to refer indistinctly to a storage device or to a distributed storage node. A serial configuration requires all units for a proper operation. In contrast, a parallel configuration is made with redundant units and requires only one unit to operate correctly. The purpose of the block diagram is to determine all possible paths that connect the input to the output. In our case, an input can be interpreted as a block read request and the output as the requested block.

We define the total system reliability for mission time T as the probability that the system succeeds $R(T) = P\{System \leftrightarrow Succeeds\}$. If the system has many components the formula is expressed using the probability of success of each element $R(T) = P_s(X_1$ and X_2 and $\ldots X_n > T)$. The probability that a system fails is derived from the equations above, $P_f\{System \leftrightarrow Fails\} = 1 - P_s\{System \leftrightarrow Succeeds\}$. To simplify the notation, we will assume that the mission time is T=1 and omit the time reference in the formulas. Ultimately, the common metric to refer to reliability is with the numbers of nines. For instance, if the estimation of reliability for mission time unit yields R=99.999%, we say the system has *five nines* reliability.

The probability that a disk node i provides the requested block is $P(x_i)$, and the probability that it fails to satisfy the request is $P(\bar{x}_i)$. If failures are independent, the basic formulas for a serial and parallel configuration for two nodes are:

$$P_s(series) = P(x_1)P(x_2) \tag{5.2}$$

$$P_s(parallel) = 1 - P(\bar{x}_1)P(\bar{x}_2) \tag{5.3}$$

When failures are not independent, the conditional probabilities must be considered in the equations.

An entangled storage system combines storage units in series and parallel configurations, see Figure 5.8. The example illustrates the read operation of one d-block by fetching the d-block itself or by performing local repairs through pp-tuples (t_{RH}, t_{LH} or t_H). Based on Equation 5.3, the expression for the four lines parallel configuration shown in the figure becomes:

$$P_s(parallel) = 1 - P(\bar{x}_1)P(\bar{t}_{RH})P(\bar{t}_{LH})P(\bar{t}_H) \tag{5.4}$$

where $P(\bar{t}_{RH}) = P(\bar{t}_{LH}) = P(\bar{t}_H)$ are calculated using Equation (5.2).

We finally introduce the cut-set method. A cut is a set of edges such that the graph becomes disconnected if they are removed. In particular, we are interested on the minimal cut-sets, whose edges do not form subsets that are also cut-sets. This method is interesting because there are typically fewer cut-sets than tie-sets, the alternative method that takes into account the set of edges that connect the input to the output.

The basic formula for computing reliability using cut-sets is:

$$R_{in\_out} = 1 - P(C_1 + C_2 + \ldots + C_j) \tag{5.5}$$

It requires an expansion of 2^j terms (Colbourn, 1993). If the events are independent, however, Equation (5.5) can be approximated. A lower bound can be computed as (Shooman, 1990):

$$R_{in\_out} \geq 1 - P(C_1) + P(C_2) + ... + P(C_j) \tag{5.6}$$

Reliability analysis of a large and complex system may become intractable without the help of a divide-and-conquer methodology. The number of calculations necessary for this study was significantly reduced by using a hierarchical decomposition of subsystems and a combination of the equations mentioned above. Similar treatment have been proposed for instance in (Rosenthal, 1977) and (Li and Haimes, 1992).

We now indicate how we perform the system decomposition. Our starting points are the components involved in the retrieval of one d-block. We consider local and global repairs (see repairs in Chapter 4). At the local repair level, there are three tuples that can regenerate the d-block in case the disk node that contains the actual d-block fails. If various disk nodes fail, it may be necessary a global repair process. At the global level, many other nodes can provide the same content. Based on the recovery trees presented in the previous chapter, we design various reliability block diagrams. As a short reminder, a recovery tree is a binary tree that indicates the blocks that are used during a repair process and the order in which they are combined (see global repairs in Chapter 4). The root of the tree is the repair target (d-block). The leaves are the available blocks used in the recovery of the root and all inner nodes. For a certain root, many recovery trees are possible. In particular, we indicate the first pp-tuple that was used to build the tree (the elements of the tuple are the root's children). This section discusses two reliability diagrams based on recovery trees built with t_H and t_{RH} pp-tuples.

5.3.2 Model Assumptions

The computation of α-entanglement reliability depends on three factors: the recovery tree depth h considered for the global repair process, the number of independent resources d in the system (disk nodes), and the code parameters (α, s, p). Note that there is a trade-off between h and d. A small d increases the chances that blocks used in the repair process are in the same failure domain. If many blocks are stored in the same failure domain, the effective deph of the recovery tree may decrease. Finally, the code parameters determine the fault tolerance as seen in the previous section. In an ideal scenario, each block in the system is stored in a different resource and the three code parameters are large.

Data blocks are encoded using AE(3,2,5). Each d-blocks and its 3 encoded p-blocks are assigned to distinct disk nodes with a round-robin placement policy. This policy simplifies reliability computation since we know in advance which blocks are located in the same unit. We restricted the number of distinct disk nodes in the system to $d = 80$. That means that when resources are exhausted, the next incoming block in the system will share the storage location with other blocks relatively close in the lattice. This number of distinct disk nodes was not chosen arbitrarily; it is derived from the code parameter $p = 5$, $80 = 16p$ (Estrada-Galinanes

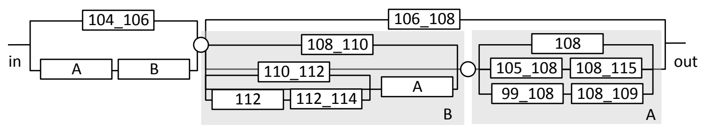


Figure 5.9 – Graph-transformation method. Reduced reliability diagram for d_{106} using t_H.

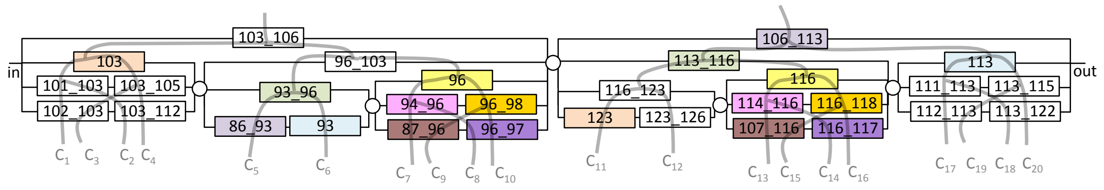


Figure 5.10 – Cut-sets method. Reliability diagram for d_{106} using t_{RH}. Distinct shades indicate an element's location: blocks with identical share are located in the same resource, whereas white blocks are stored in independent distinct units.

and Felber, 2015b). This formula ensures that we have enough blocks to perform a global repair. In agreement with the placements, we limit the tree depth to $h = 4$. The recovery trees used in this study have deliberately pruned nodes and cycles are avoided, e.g., a child node cannot be used to recover a parent node if both blocks are stored on the same disk node.

We present two reliability block diagrams based on the recoveries trees. Any path from input to output is valid to repair a d-block as long as all disk nodes that are part of the path are available. Note that the diagrams could be expanded with additional paths using different placement policies, e.g., with random placements and a larger d.

We assume that all units have the same probability of failure, with an unknown probability density function, and that disk failures are independent. For numerical evaluation, we use $P(x_i) = p = 0.95$ and $P(\bar{x}_i) = q = 0.05$ for time $t = t_1$.

5.3.3 Reliability Computation

We start with the analysis of the block diagram with t_H shown in Figure 5.9. This diagram corresponds to one of the possible recovery tree for d_{106}. In the example, the tuple t_H was used to decide the root's children.

The reliability expression is the probability of the union of all events containing a valid path to connect the input to the output. For example, one valid path to generate d_{106} at the output is $\{p_{104,106}, p_{108,110}, d_{108}\}$.

We assume that all components used in the repair process reside on distinct disk nodes. However, all disk nodes are considered homogeneous and with identical probability of success. This second assumption allows us to reduce the diagram and perform fewer calculations.

We opt for graph transformations (Rozenberg, 1997) to divide the diagram structure into smaller subsystems. Hence, the structure is reduced using combinations of serial and parallel transformations. As a result, it becomes simpler to compute the reliability for each subsystem and combine their contributions to the final reliability expression. The figure emphasizes subsystems in color grey. We reuse the calculus for that subsystem when it appears in other parts of the diagram. The contribution of subsystem A (from Figure 5.9) is:

$$R_A = 1 - (1-p)(1-p^2)^2 \tag{5.7}$$

Expansion of Equation (5.7) yields:

$$R_A = p^5 - p^4 - 2p^3 + 2p^2 + p \tag{5.8}$$

By carefully applying the formulas for serial and parallel configurations, the reliability offered by the H strand is:

$$\begin{aligned} R_H = p^2((p-1)p \\ (p^8 - 3p^7 + 7p^5 - 4p^4 - 4p^3 + 2p^2 + p + 1) \\ ((p-1)p(p^2-2)+1)-1)^2 \end{aligned} \tag{5.9}$$

One can finally numerically evaluate reliability. For instance, for $p = 0.95$, it yields:

$$R_H \approx 0.999926 \tag{5.10}$$

The block diagram based on t_{RH} (Figure 5.10) and its counterpart based on t_{LH} (omitted) require judgement due to the placement policies. The helical RH- and LH-strands connect blocks that are more distant in time (and in the lattice). In our environment, we guarantee that a section of the lattice, which contains 80 blocks, resides on independent disk nodes. Therefore some blocks will inevitably be stored on the same disk node, hence reducing the lattice resilience.

We use the cut-set method to analyse the block diagram based on t_{RH}. By using a different method, we can validate previous results. Table 5.1 enumerates all minimal cut sets.

Cut Sets	Term
C_1: $S_{103\_106}, F_{103}, S_{101\_103}, S_{102\_103}$	q^4
C_2: $S_{103\_106}, F_{103}, S_{101\_103}, S_{103\_112}$	q^4
C_3: $S_{103\_106}, F_{103}, S_{103\_105}, S_{102\_103}$	q^4
C_4: $S_{103\_106}, F_{103}, S_{103\_105}, S_{103\_112}$	q^4
C_5: $S_{103\_106}, S_{96\_103}, S_{93\_96}, S_{86\_93}$	q^4
C_6: $S_{103\_106}, S_{96\_103}, S_{93\_96}, F_{93}$	q^4
C_7: $S_{103\_106}, S_{96\_103}, F_{96}, S_{94\_96}, S_{87\_96}$	q^5
C_8: $S_{103\_106}, S_{96\_103}, F_{96}, S_{94\_96}, S_{96\_97}$	q^5
C_9: $S_{103\_106}, S_{96\_103}, F_{96}, S_{96\_98}, S_{87\_96}$	q^5
C_{10}: $S_{103\_106}, S_{96\_103}, F_{96}, S_{96\_98}, S_{96\_97}$	q^5
C_{11}: $S_{106\_113}, S_{113\_116}, S_{116\_123}, F_{123}$	q^4
C_{12}: $S_{106\_113}, S_{113\_116}, S_{116\_123}, S_{123\_126}$	q^4
C_{13}: $S_{106\_113}, S_{113\_116}, F_{116}, S_{114\_116}, S_{107\_116}$	q^5
C_{14}: $S_{106\_113}, S_{113\_116}, F_{116}, S_{114\_116}, S_{116\_117}$	q^5
C_{15}: $S_{106\_113}, S_{113\_116}, F_{116}, S_{116\_118}, S_{107\_116}$	q^5
C_{16}: $S_{106\_113}, S_{113\_116}, F_{116}, S_{116\_118}, S_{116\_117}$	q^5
C_{17}: $S_{106\_113}, F_{113}, S_{111\_113}, S_{112\_113}$	q^4
C_{18}: $S_{106\_113}, F_{113}, S_{111\_113}, S_{113\_122}$	q^4
C_{19}: $S_{106\_113}, F_{113}, S_{113\_115}, S_{112\_113}$	q^4
C_{20}: $S_{106\_113}, F_{113}, S_{113\_115}, S_{113\_122}$	q^4

Table 5.1 – Minimal cut sets of the structure shown in Figure 5.10.

The reliability lower bound using Equation (5.6) is:

$$R_{RH} = R_{LH} \geq 1 - (8q^5 + 12q^4) \tag{5.11}$$

Substitution yields:

$$R_{RH} = R_{LH} \geq 0.999922 \tag{5.12}$$

As expected, the difference between both methods is minimal. We simply use Equation (5.11) to compute an approximation for the final expression that considers the d-block itself and the three diagrams built on recovery trees:

$$R_{in\_out} \geq 1 - q(8q^5 + 12q^4)^3 \tag{5.13}$$

By using a value of $q = 0.05$, we obtain a reliability of 13 nines.

	R_{in_out}	With $p = 0.95$, $q = 0.05$
AE(3,2,5)	$\approx 1 - q(8q^5 + 12q^4)^3$	13 nines
4-replication	$1 - q^4$	5 nines
$(4, 12)$ RS	$\sum_{k=r}^{n} \binom{n}{r} p^r (1-p)^{n-r}$	14 nines

Table 5.2 – Comparison of reliabilities for various erasure codes.

5.3.4 Brief Discussion

The relevance of our analytical results is reflected when we draw a comparison with other erasure codes. Replication is basically a parallel configuration and Reed-Solomon is a r-out-of-n configuration. Table 5.2 provides values for some specific configurations.

In our estimations we assume a restricted environment of only 80 storage devices. Therefore, the calculations lead to a conservative estimation of the reliability and we can expect even higher resilience when using more resources and deeper recovery trees.

5.4 Summary

The analysis of fault-tolerance gave us a good idea of the impact of code parameters on the size of failure patterns. This chapter have shown the benefits of redundancy propagation as well as established its boundaries. A larger α implies a larger node degree and that is translated in more node connectivity (multiple paths). A larger s and/or p impacts on the length of the path between two endpoint dead nodes. These last two parameters permit increasing faul tolerance without generating more storage overhead (only α impacts in the number of parities per data block) or increasing the repair cost of single failures (any code setting repairs single failures with two blocks). Our qualitative study establishes the boundaries for redundancy propagation. Generalizing our knowledge on failure patterns is difficult, e.g., there are critical patterns that for certain settings are not the smallest pattern but become the smallest when the code parameter changes. A quantitative approach is complementary to our study. One approach could be *brute force* analysis, however the search space grows exponentially with the code parameters. That being said, it could allow us to better understand the behavior of the codes.

The assessment of entanglement codes is concluded with a reliability analysis based on combinatorial methods. We applied two methods graph transformations and cut-set to verify the results. This study permits the comparison with other codes from the theoretical perspective. Our study undersells entanglement codes. In our settings entanglement codes achieve 13 nines and Reed-Solomon 14 nines. However, data survivability was superior for entanglement codes when we did practical experiences and simulations in the previous chapter. That shows the important role of placements and the number of failure domains considered in the study. In the practical experiments the assumption was that every block was stored in a different location. In the simulations, 1 million of d-blocks and their corresponding p-blocks were distributed in 100 distinct locations using random placements. Here, the placement policy was round-robin, blocks were distributed with a round-robin policy and the length of recovery trees was pruned.

6 Beyond Entanglement Codes

Entanglement codes propagate redundant information across storage locations in a centralized or decentralized system to provide multiple paths to recover unavailable content. The entanglement can be a simple chain of alternated unencoded data blocks and parity blocks or become more complex as in the case of helical entanglement codes, in which each data block participates in three chains. In that sense, adding chains of entangled data has the same cost of adding a replica for each data blocks that is part of the chain. For some applications, this level of replication may be too expensive. Spigot codes are built on top of entanglement codes to increase their code rate and reduce the space footprint from entangled data. We study two techniques based on code puncturing and XOR coding and present four spigot codes. This chapter is largely based on recent submitted work (Estrada-Galinanes et al., 2017a).

6.1 Introduction and Motivation

Data entanglements are a mechanism to propagate redundant information across a large scale distributed system and create a virtual storage layer of highly interconnected elements. Simple entanglement chains (Estrada-Galinanes et al., 2016) require equal numbers of data and parity drives; therefore, they have the same space overhead as mirroring. Complex entanglements can be defined by combining multiple simple entanglement chains into a mesh, as in the case of helical entanglement codes (Estrada-Galinanes and Felber, 2013).

Erasure codes based on data entanglement are a promising solution for archiving data since they provide reasonable trade-offs between security, resource usage and performance. A unique feature of entanglement codes is that propagation can be used to improve fault-tolerance without increasing the storage overhead or the repair cost of the code. In that sense, the storage overhead is limited by the number of entanglement chains in which a data block participates. Participation means to contribute with a parity block that enlarges the chain. For example, in the case of helical entanglement codes, each data block participates in three chains, therefore the storage system requires three times more capacity to store the redundant data. Even though helical entanglement codes can be tuned to provide extremely high fault-

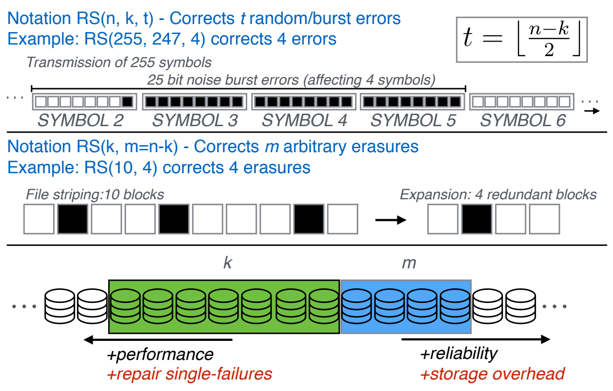

Notation RS(n, k, t) - Corrects t random/burst errors
Example: RS(255, 247, 4) corrects 4 errors
$t = \left\lfloor \frac{n-k}{2} \right\rfloor$
Transmission of 255 symbols
25 bit noise burst errors (affecting 4 symbols)
SYMBOL 2
SYMBOL 3
SYMBOL 4
SYMBOL 5
SYMBOL 6
Notation RS(k, m=n-k) - Corrects m arbitrary erasures
Example: RS(10, 4) corrects 4 erasures
File striping:10 blocks
Expansion: 4 redundant blocks
k
m
+performance
+repair single-failures
+reliability
+storage overhead

Figure 6.1 – Understanding different notations for Reed-Solomon codes and the limitations for valid “k” and “m” in practical solutions for storage systems (usually $k + m \leq 20$). Repair blocks in settings with large values for “k” and “m” is cumbersome when blocks are distributed in many storage devices.

tolerance, the “entry space requirements” are expensive for some applications. For instance, at the time of this writing, the Internet Archive (Jaffe and Kirkpatrick, 2009) has only two mirrors: the official mirror at archive.org and the second mirror at Bibliotheca Alexandrina. The organization estimates that building a third partial mirror in Canada will cost millions of dollars (Brewster, 2016).

The motivation behind the design of spigot codes is to reduce the space footprint from entangled data. This chapter proposes two methods for constructing spigot codes on top of entanglement codes. A first approach is puncturing, a standard technique used in coding theory in which, after encoding, some of the encoded symbols are not stored in the system. A second approach is compaction, which reduces the number of encoded symbols by combining them. Both approaches are not exclusive and can be combined in different ways to trade space with fault tolerance and repair cost. We study various spigot codes constructions.

6.2 Background

Rateless codes permit the variation of the block length to adapt transmission according to the channel’s noise level. The first practical known construction is the Luby transform (LT) codes (Luby, 2002). Raptor codes are the first implementation with linear time complexity for encoding and decoding (Shokrollahi, 2006). Rateless codes are usually compared with a water fountain that can supply an infinite number of drops, each of size l bits, a metaphor for encoded packets. If a message of size Kl is encoded with this water fountain, it will be necessary to collect in a bucket a number of drops a bit larger than K for decoding the message (MacKay, 2005). One application is a data carrousel that streams a file continuously during a certain time. Rateless codes are sparse-graph codes for channels with erasures. Channels with erasures are relevant for Internet packet transmissions as well as for distributed storage. Two versions of Raptor codes became a Request for Comment Standard of the Internet

Engineering Task Force: the R10 (Shokrollahi et al., 2007) and the RFC6330 for RaptorQ (Luby et al., 2011). The standards fix the number of maximum source symbols to 8,192 for R10 and 56,403 for RaptorQ. This limitation is related to the problem of finding an optimal degree distribution that assures the successful decoding of the symbols.

On the other side of the spectrum are *maximum distance separable (MDS) codes* that achieve optimal rate. A linear code is said to be an (n, k, d) MDS code if it takes as input words of length k to generate codewords of length n and d is the minimum number of bits that separates two distinct codewords. Reed-Solomon codes, (k, m)-RS codes, are a remarkable class of MDS codes. They tolerate $m = n - k$ arbitrary erasure patterns. During encoding, a file is striped in k blocks and expanded with m extra blocks. The file is recovered with any k blocks out of $k + m$. The code rate $k/(k + m)$ indicates the proportion between the amount of real information and the total amount of symbols transmitted on the channel.

One of the goals in the design of entanglement codes was to relax the constrains behind a valid (k, m) setting that is practical for storage applications. An ideal code would have flexibility in its parameter values so the storage system can handle different workloads, heterogeneous storage devices, and different file sizes efficiently. Furthermore, it would have an encoding and decoding algorithm that does not depend on code constructions with a specific valid degree distribution. To elaborate a bit more about the current constrains for (k, m) settings we need to refer the reader to the different applications of RS codes. Figure 6.1 provides examples of RS codes applications as well as illustrates the different notations for RS settings and the meaning behind it. RS codes are defined over Galois Fields denoted as $GF(2^w)$ that have $n = 2^w - 1$ words or code symbols of w bits length and $2t$ out of the total words are used as parity or check words. For example, a code defined over $GF(2^8)$ has 255 words of 8 bits length (1 byte). The top image presents the use of RS to tolerate random and burst errors in data transmission and for writing bytes to one storage device. The setting RS(n=255,k=247,t=4) can protect 247 symbols since the decoder can correct up to 4 symbols in which the position is not known. The example shows how a burst error of 25 consecutive bits that fit inside 4 symbols can be repaired with RS codes. The middle image illustrates the typical setting in RAID-like systems (Plank et al., 1997). Blocks are distributed in different storage devices. The failure model is different from the previous case because the position of the error is usually known. Hence, we talk about erasures and, in this case, the decoder can correct up to $m = n - k$ erasures. In RAID-like systems, the object to store, e.g., a file, is striped into k blocks and expanded to m blocks. That brings a simplification and a second meaning in the code notation, e.g., the setting RS(k=10,m=4) indicates that the system distributes the blocks in a *stripe* of $k + m = 14$ storage devices. When decoding is done in a distributed environment the repair cost associated to the bandwidth and I/O operations constrains RS settings to practical small values. Other codes similar or based on RS codes inherit the same limitation. However, large values for “k” and “m” can increase performance and reliability, which is highly beneficial for many storage applications.

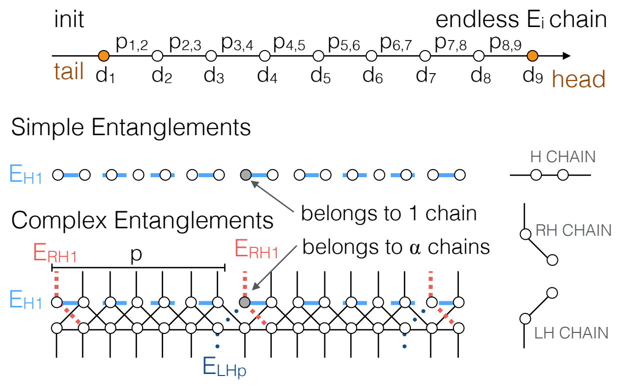


Figure 6.2 – An entanglement chain E_i alternates d-blocks and p-blocks elements. Simple entanglements (SE) are defined by a single horizontal chain. Complex entanglements (CE) defined a lattice composed by multiple horizontal and helical chains. The example corresponds to a 7-HEC lattice composed with 7 pairs of E_{RH} and E_{LH} chains.

6.3 Entanglement Chains, Benefits and Limitations

The foundations of entanglement codes are entanglement chains or strands and the two core elements are data blocks, *d-blocks* and parity blocks, *p-blocks*. The entanglement chains alternate *d-blocks*, and *p-blocks* that are represented by a node and an edge, respectively. The chain reveals temporal relationship between blocks since the most recent blocks are always written at the head. The encoder builds chains by computing the exclusive-or (XOR) of two consecutive blocks at the head of a strand and inserts the output adjacent to the last block. Since unencoded data blocks are stored in the system there is no need for decoding in the absence of failures. Figure 6.2 illustrates an entanglement chain and the simple and complex entanglements that are built with it. A p-block of the chain, e.g. $p_{4,5}$, is the result of XORing the previous two elements: $p_{4,5} = d_4 \oplus p_{4,5}$. Later, the same p-block could be repaired by performing the same operation or by XORing the next two elements: $p_{4,5} = d_5 \oplus p_{5,6}$. A d-block could be repaired by XORing the two adjacent elements, example: $d_5 = p_{4,5} \oplus p_{5,6}$. Simple entanglements are constructions of single chains. Multiple simple entanglements can coexist in a storage system but any d-block belongs to only one chain. Complex entanglements combine horizontal and helical (right-handed and left-handed) chains into a lattice. The purpose of combining multiple chains in a lattice is to increase exponentially the number of paths that the repair algorithm can take to repair d-blocks. The example shows the 7-HEC lattice composed by two horizontal chains and $p = 7$ double helix strands (chains) built with a helical entanglement code. Even though the total number of chains is $2 \cdot 7 + 2 = 16$ chains, each d-block belongs to one horizontal chain and a double helix chain. Note that a p-HEC lattice is equivalent to a lattice built with alpha entanglements AE(3,2,p), where $p \geq 2$. Since most of the studies conducted in this chapter only vary p, we prefer the p-HEC notation as it is more compact.

Entanglement codes are designed to relax the constraints behind practical (k, m) settings for storage systems. The entanglement encoder creates a virtual storage layer of highly interconnected elements (encoded packets). Then, elements can be mapped to storage devices using different placement algorithms. The virtual storage layer grows as more elements are encoded and integrated in a mesh of entangled blocks. In this context, the traditional code parameters k and m are relaxed bringing important benefits, specifically:

1. There is no overhead when striping a small file in a fixed k value because any d-block is encoded individually.
2. Blocks can be distributed across a larger number of storage devices that can be accessed in parallel to improve throughput.
3. The repair cost of single blocks is extremely low. Single failure repair is done by the computation of a bitwise XOR operation on two blocks.
4. The entanglement parameters define the interdependencies between elements and how redundancies are propagated. As a result, the parameters can improve fault tolerance without requiring more storage overhead. In many disaster scenarios, entanglement codes outperform RS codes.

A caveat is that Equation 4.2 tell us that the code rate is computed as

$$\frac{1}{\alpha + 1},$$

where α is the number of chains that each block belongs to. In other words, the system stores α p-blocks and 1 d-block each time that a d-block is encoded . The storage overhead that is caused by inserting d-blocks in one additional chain is 100%. That means that storage overhead does not grow gradually in the way that (k, m) settings allow. Instead, increasing α means increasing storage overhead in steps of 100%, or the equivalent of adding more replicas in a replicated system. Simple entanglements, $\alpha = 1$, are equivalent to mirroring, helical entanglement codes, $\alpha = 3$, are equivalent to a four-way replicated system. This “entry space requirement” is expensive for some applications.

6.4 Spigot Codes

Spigot codes are built on top of entanglement codes. Using two techniques we can reduce the space footprint from entangled data. Basically, spigot codes trade space for time because less space overhead will impact the repair time and the length of time during which the system will not suffer data loss. The repair algorithm of entanglement codes recovers data in one or more rounds, depending on the failure pattern. While the repair approach could be iterative or recursive, if multiple blocks are missing, it may be necessary to repair some blocks first. It is reasonably to think that less redundancy means that the number of rounds required for full repair will increase. Additionally, we expect that a system without maintenance degrades faster when fewer redundant blocks are available. Finally, spigot codes allow to trade off space for time temporarily. It is possible to get back to the full benefits of entanglement codes without the need to recode all data.

We describe the two techniques used in spigot codes:

Puncturing. It is a standard technique use in coding theory in which, after encoding, some of the redundant symbols are deleted to reduce the blocklength of the code (Ha and McLaughlin, 2003; Pishro-Nik and Fekri, 2007). This technique can be easily applied to d-blocks and/or p-blocks. We define three codes: *spigot(-,d,EC)*, *spigot(-,p,κ,EC)* and *spigot(-,p,λ,EC)* with EC being the entanglement code that is the supplier (mother code) for the spigot. The principle to construct the spigot is to eliminate blocks; therefore, all spigots are identified with "-". Decoding spigot codes is equivalent to decoding entanglement codes.
XOR Coding. This method aggregates parities from various strands by XORing two or more parities together and keeping only the resulting new block. The notation for spigot codes built using XOR coding is *spigot(+,dd,σ,EC)*. The principle to construct the spigot is to aggregate blocks; therefore, all spigots are identified with "+". Decoding spigot codes requires to decode first the aggregated blocks, followed by a normal decoding for entanglement codes.

6.4.1 Spigot(-, d, EC)

The most trivial way of puncturing entanglement codes is deleting all d-blocks and only keep the p-blocks. The code rate corresponds to Equation 4.3

$$\frac{1}{\alpha}.$$

The total space savings is the result of dividing the new code rate with the original code rate for entanglement codes. In helical entanglement codes, which are defined for $\alpha = 3$, the total space savings is 25%. Deleting d-blocks implies that the code becomes non-systematic and every read will have the overhead of decoding the d-blocks. Therefore, this option may not be the best option for read intensive applications. Placements considerations may reduce the overhead. For instance if all the p-blocks belonging to the same strand are stored sequentially in the same device, the two p-blocks necessary to regenerate one d-block would be accessed faster. The notation *spigot(-,d,EC)* indicates “-” for puncturing, “d” since the target of puncturing are d-blocks, and EC is the inner entanglement code used to supply the spigot, for example *spigot(-,d,5-HEC)* is equivalent to a spigot code built using 5-HEC to encode d-blocks and delete them afterwards.

6.4.2 Spigot(-, p, κ, EC)

In this code, puncturing is applied to p-blocks on the horizontal strands. We define κ as the puncturing factor, which is any integer number in the range of $[0, 2^p - 1]$, where p is one of

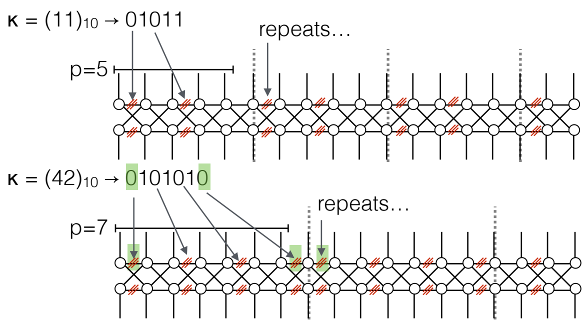


Figure 6.3 – The edges (p-blocks) from horizontal chains marked with colored strides are deleted according to κ. The examples shown correspond to spigot(-,p,11,5-HEC) and spigot(-,p,42,7-HEC).

the parameters of entanglement codes. For example, when the supplier code for the spigot is 5-HEC, $\kappa \in [0,31]$. Learning which p-blocks are deleted is straightforward from the binary representation of κ as seen in Figure 6.3. The puncturing factor determines which p-blocks are deleted in a sequence of p p-blocks on the horizontal strand. To illustrate, one of the examples in the figure corresponds to the spigot(-,p,42,7-HEC). The binary form of $\kappa = (42)_{10}$ is 0101010. The position of zero digits indicate the parities that are removed. The pattern is repeated to the right until the end of the lattice. The same pattern is mirrored on both horizontal strands.

The achieved reduction for a certain κ is derived from calculating the Hamming weight W of the factor κ written in binary format. The Hamming weight is the distance of κ to the zero word, which corresponds to the number of “1” digits in the binary form of κ, eg. $W(\kappa) = 3$ for $\kappa = 42$.

The code rate computed with Equation 4.2 is modified to

$$\frac{1}{\alpha + 1 - \frac{1-W(\kappa)}{p}} \,. \tag{6.1}$$

The storage savings are less than spigot(-,d,code) unless $W(\kappa) = p$ (which corresponds to $\kappa = 2^p - 1$). The total space saving in spigot(-,p,20,5-HEC) is 9% and approximately 14.3% in spigot(-,p,85,7-HEC).

Puncturing can be applied simultaneously to d-blocks and p-blocks to achieve more significant savings. For instance, the total space savings in spigot(-,**dp**,20,5-HEC) would be 34% and approximately 39.3% for spigot(-,**dp**,85,7-HEC) .

6.4.3 Spigot(-, p, λ, EC)

In this code, puncturing is applied to p-blocks on helical chains. We define the vector $\lambda = [\lambda_1, \lambda_2, \cdots, \lambda_p]$ where its i-th element is the correspondant puncturing factor for helical chains

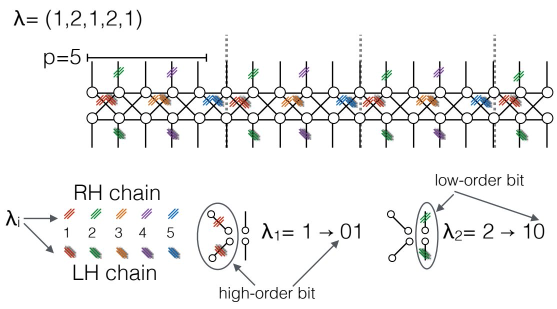


Figure 6.4 – Spigot(-,p,(1,2,1,2,1),5-HEC): p-blocks from helical chains are deleted according $\lambda = (1,2,1,2,1)$. The convention to identify which edges are pruned is to start counting on the edges that are at the right side of the node in a p section of the lattice and then continue on the same strand as indicated in the figure.

E_{RHi} and E_{LHi}. The puncturing factor is any integer number in the range of $[0, 2^s - 1]$, where s is an entanglement parameter that indicates the number of horizontal chains. If the supplier code is 5-HEC, λ_i is in the range of $[0,3]$. In fact, all spigots that use a supplier code p-HEC have $\lambda_i \in [0,3]$ as they are built with two horizontal strands ($s = 2$) but other type of entanglements may use more strands. Learning which p-blocks are deleted is straightforward from the binary representation of λ as seen in Figure 6.4. For example, the puncturing factor $\lambda_1 = 1$ written in binary form *01* determines which p-blocks are deleted in a sequence of 2 p-blocks on E_{RH1} and E_{LH1}. The puncturing pattern is repeated until the end of the chain.

The code rate computed with Equation 4.2 is modified to

$$\frac{1}{2 + (\alpha - 1)(1 - \sum_{n=1}^{p} \frac{W(\lambda_i)}{s \cdot p})}, \tag{6.2}$$

for entanglement codes with $\alpha = 2$ or $\alpha = 3$. An entanglement made with $\alpha = 2$ is similar to $\alpha = 3$ but with only one helical chain. Simple entanglements ($\alpha = 1$) do not have helical chains. More complex entanglements need a per case study since they may require different puncturing factors. The second term of the denominator calculates the storage savings in each of the $(\alpha - 1) \cdot p$ helical chains

A slight variation would be to apply λ to only one type of helical chains (RH or LH). For this case, we use the notation spigot(-, p, λ', EC). Equation 6.2 should be modified accordingly.

6.4.4 Spigot(+, pp, σ, EC)

This code uses a XOR coding scheme with some similarity to the one proposed for Facebook's f4 storage system (Muralidhar et al., 2014). In their case, they use the scheme to reduce storage requirements while providing enough redundancy to survive the failure of one datacenter. They achieve their goal by keeping the result of XORing blocks from two different volumes located at different locations in a third location. We do not discuss location since mapping

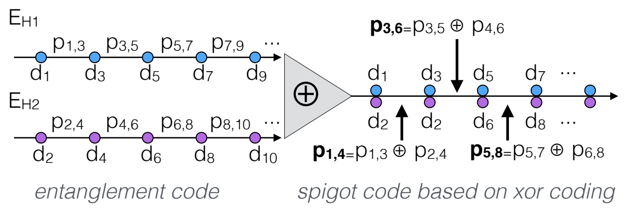


Figure 6.5 – Spigot(+,pp,2,SE): Two simple entanglement chains are aggregated into a single chain to save storage space. Odd and even labels for d-blocks help to identify their source chain.

blocks into the physical storage layer depends on the use case application.

Our scheme reduces two or more horizontal chains into a single chain (with multiple d-blocks) as indicated in the example of Figure 6.5. The σ coefficient indicates how many horizontal chains are aggregated into the stored chain. Its value is any factor of s, greater than one, for reductions that affect two or more chains from the same lattice. In the example, the spigot code keeps the same number of d-blocks but cuts the number of p-blocks in half. The scheme could also be applied to mutiple simple entanglements chains. In such case, the σ coefficient indicates how many simple chains are involved and its value is any number greater than one.

The chain of an spigot code based on a XOR scheme alternates σ d-blocks and p-blocks, where the p-blocks aggregate the information of σ p-blocks. That means that single blocks are repaired with $\sigma+1$ blocks instead of two blocks as in normal entanglement chain. For example, the repair of d-block d_5 from the spigot(+,pp,2,SE) shown in Figure 6.5 is done with $d_5 = d_6 \oplus p_{3,6} \oplus p_{5,8}$. Similarly, the d-block d_6 is repaired with $d_6 = d_5 \oplus p_{3,6} \oplus p_{5,8}$. Parity blocks still have two possible ways of repairing, but the equations include $\sigma+1$ terms. For example, the repair of p-block $p_{3,6}$ is done with $p_{3,6} = d_2 \oplus d_3 \oplus p_{1,4}$ and $p_{3,6} = d_5 \oplus d_6 \oplus p_{5,8}$.

The code rate computed with Equation 4.2 is modified to

$$\frac{1}{\alpha + \frac{1}{\sigma}} \,. \tag{6.3}$$

That means that the spigot(+,dd,2,5-HEC) achieves 12.5% of total space savings. In the case of three simple entanglements, spigot(+,dd,3,SE), the space savings are 37.5%. For more savings, a cross-lattice XOR coding scheme is possible. The code rate computed with Equation 4.2 for cross-lattice is modified to

$$\frac{1}{\alpha + \frac{1}{l \cdot \sigma}} \,, \tag{6.4}$$

with l being the number of lattices. For instance a cross-lattice $l = 2$ with spigot(+,dd,2,5-HEC) reduces the total four horizontal chains of two 5-HEC lattices into one and achieves 18.75% of total space savings.

Entanglement Codes	**SE** $\alpha = 1$	**5-HEC** $\alpha = 3$
Extra storage	100%	300%
Single failures	2	2
Spigot Codes	**(+,pp,2,SE)**	**(+,pp,2,5-HEC)**
XOR Code	$\sigma = 2$	$\sigma = 2$
Extra storage	50%	250%
Single failures	3	2
Spigot Codes	**(-,d,5-HEC)**	**(-,p,λ',5-HEC)**
Puncturing		$\lambda' = (2,2,2,2,2)$
Extra storage	200%	200%
Single failures	2	2

Table 6.1 – Selected entanglement and spigot codes.

Finally, spigot(+,pp,σ,EC) can be used in combination with puncturing p-blocks on helical strands.

6.5 Evaluation

To evaluate spigots we compare them with their supplier code to see what is the price to pay for a higher rate code. We characterize how spigot codes perform after a catastrophic failure and we measure data loss after repairs. With fewer redundant blocks the repair algorithm may need more time to repair data. Thus, we study how many additional rounds the repair of an spigot code needs.

Our framework lets us observe what happens with data in environments that either accumulate plenty of failures before repairs take place or a large number of failures happen all at once. All d-blocks and p-blocks get a random number from 0 to 99 that represents the storage location. A disaster is simulated by changing the availability of a certain number of locations and trying to repair the missing data blocks. These figures are given in percentages (10-50%).

Repair single failures and storage overhead. To increase the code rate of the mentioned entanglement codes we used spigot codes according to Table 6.1. Our baseline are simple entanglement codes SE ($\alpha = 1$) and complex entanglement codes built with helical entanglement codes: a 5-HEC setting with $\alpha = 3$. The spigot(+,pp,2,SE) achieves a total space saving of 25% with respect to the 2x storage requirement of SE codes. The spigot(+,pp,2,5-HEC) achieves a total space saving of 12.5% with respect to the 4x storage requirement of 5-HEC codes. The spigot(-,d,5-HEC) and spigot(-,d,λ',5-HEC) achieve each a total space saving of 25% with respect to the 4x storage requirement. The cost of repairing single failures rises to three blocks (instead of two) only for spigot(+,pp,2,SE). In other codes there are other chains with enough p-blocks to guarantee single failure repairs using two blocks.

Data loss and repair rounds. The repair algorithm proceeds in rounds, in each of which more d-blocks and p-blocks are recovered. It stops when is not possible to keep recovering missing blocks. Figure 6.6 illustrates data loss for the entanglement codes and spigot codes

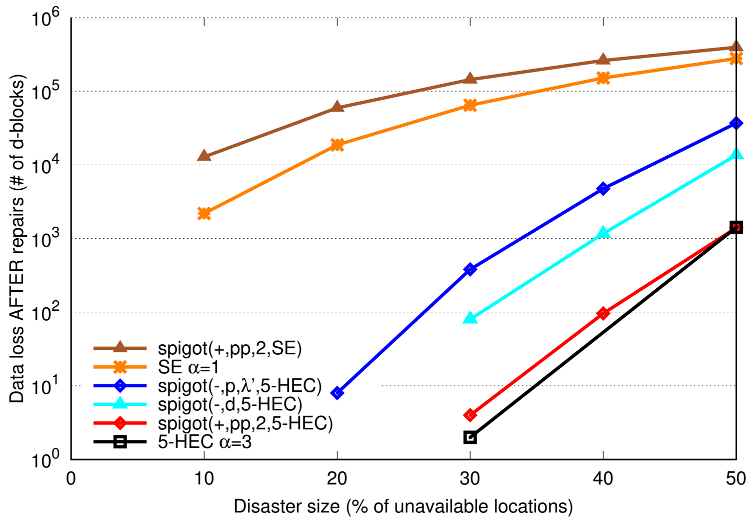


Figure 6.6 – d-blocks that could not be repaired.

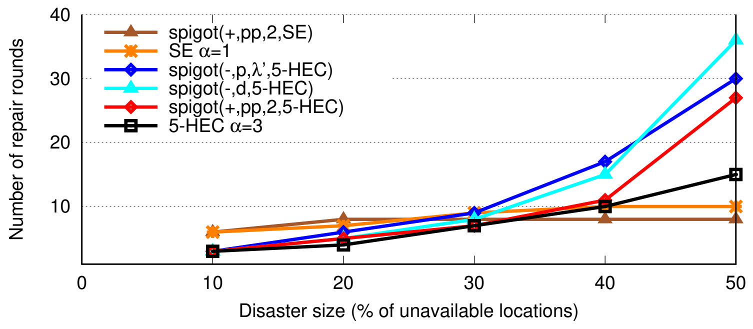


Figure 6.7 – Number of repair rounds until data loss.

of Table 6.1. An interesting observation is that spigot(+,pp,2,5-HEC) saves 12.5% storage and performs very well in catastrophic disasters, very close to its source code 5-HEC. SE and its spigot(+,pp,2,SE) do not perform well in large disasters without maintenance but it may be worth to study its spigot in more detail to replace mirroring in applications that have some maintenance. The differences between both spigot codes based on puncturing suggest that it is preferable to delete d-blocks instead of p-blocks for maintaining a higher reliability while reducing space. That comes at the price of a non-symmetric code that requires decoding each d-block. Figure 6.7 gives an idea of how the number of rounds increases with less redundancy and larger disasters.

6.6 Summary

Our study shows that code puncturing and XOR encoding are good techniques to increase the code rate of entanglement codes. Spigot codes can be combined to achieve considerable space savings. In this chapter we presented examples that reduce up to 50% of the storage requirements of vanilla entanglement codes. As seen in the study of repair rounds, our examples don't increase the repair overhead too much until large scale disaster that affect 30% or for some cases 40% storage locations. The simulations show the flexibility of entanglements codes and their spin-off codes. More studies are needed to find efficient combinations in each particular application.

7 Towards Entangled Storage Systems

This chapter is intended to be a guideline for implementing entanglement codes. We present some use cases and discuss domains in which entanglement codes could be applied. Alpha entanglement codes are not restricted to a particular storage architecture. In particular we present two use cases: the first case is a cooperative storage system built on top of a decentralized database, the second example showcases a centralized storage architecture built using disk arrays. We finish this chapter by discussing practical aspects of entanglement codes and lessons learned during the prototyping of our ideas. This chapter discussed topics related with the implementation of entanglement codes based on preliminary ideas presented in FAST'15 and '16 Work in Progress Sessions talks (Estrada-Galinanes and Felber, 2015a; Estrada-Galinanes and Felber, 2016) and Eurosys Doctoral Workshop (Estrada-Galinanes, 2016) and technical reports for my doctoral program (Estrada Galinanes, 2013; Estrada Galinanes, 2015).

7.1 Use Case: A Geo-Replicated Backup System

In this example a community of users create a cooperative storage network to share storage and bandwidth resources. Users may need to back up different amounts of data. They keep their own data in their local nodes and upload redundant information to geographically distributed nodes. The storage capacity used by each user has to be agreed at the beginning and renegotiated as needed. The system is a two-tiered architecture that aims at supporting efficient data protection and integrity. The lower tier is composed of *storage nodes* that share space to store parity blocks from other users. The upper tier consists of *broker nodes* that entangle and disentangle data. In this example a single node has both roles. The nodes are organized in a loosely connected cluster. In this context, the term "node" is used to refer to the mentioned nodes. The text clarifies if nodes refers to vertices in the entanglement lattice graph. To avoid confusions, we use the term *d-blocks* and since lattice's edges represent parity blocks, we use the term *p-blocks*. It is worth noticing that the design of cooperative storage requires consideration of various types of attacks such as free riding abuses that are beyond the scope of this work. In particular, the problem of free ride in the context of a collaborative internet backup was studied by other authors (Lillibridge et al., 2003).

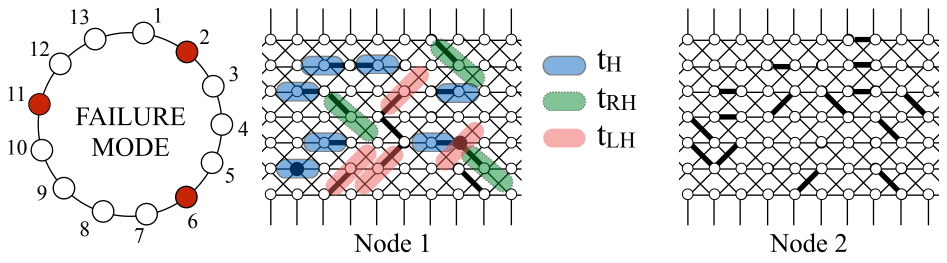


Figure 7.1 – Failure mode when three storage nodes fail. Lattices shows different degradation at each node. The repair process at node 1 uses the tuples indicated with different shades.

7.1.1 Redundancy Scheme

Each user manages his own entanglement lattice via the broker, hence multiple lattices coexist in the system. Moreover, the system could keep lattices with different settings. In the simplest scenario, the broker is a service running in the user's machine. In more complex scenarios, the broker could be a super node that encodes data for a group of users. When a user wants to backup a document, he sends the file to the broker. The broker splits the file into *d-blocks* and encodes them. To encode data, it needs to fetch some parities from other storage nodes, or keep some parities in memory for performance. Blocks are mapped to the storage nodes using a deterministic or random placement algorithm. A unique key that identifies the block is used to locate the block. In the next section we will discuss more about this key. In a failure-free environment, users can access their data directly from their local machines. In other words, the broker does not need to decode (disentangle) data.

Memory Requirements. When parities are kept in memory, the memory footprint of the broker is linear in the number of distinct strands that create the entanglement lattice. For instance, the code setting AE(3,2,5) requires to keep in memory the last 12 generated parities. More specifically, the memory requirements are related to the total number of strands in the system (see Equation 4.1). This is because the new parities are added to the head of this strands.

Recovery when a Broker Crashes. If the broker crashes, the last recent parities kept in memory will be lost. Then, they broker must retrieve them again from other storage nodes.

7.1.2 Failure Mode

For this application, the two events that trigger data decoding are when users do not have access to their own locally stored d-blocks or when their lattices deteriorate due to other nodes failures. How node failures are detected and notified is not discussed here but there is abundant literature on this topic (Rhea et al., 2004; Lakshman and Malik, 2010; Huan and Nakazato, 2015). Figure 7.1 shows a scenario in which three nodes are unavailable and illustrates how the incident impacts on multiple elements in distinct lattices. The size and

Steps	Example
1. Obtain dp-tuple id:	$\{key_{21}, key_{16,21}\}$,$\{key_{26}, key_{26,31}\}$
2. Choose p-block id:	$key_{16,21}$
3. Compute location key:	n_5
4. Get block:	$p_{16,21}$
5. Repair block:	$p_{21,26}$

Table 7.1 – Operations involved during the repairs.

characteristics of the incident dictate the patterns formed on the lattices. Parity block repair is automatically distributed, assuming that all users will be interested in the regeneration of their lattices to maintain the same level of redundancy for their data. If a node is not able to repair the lattice, other nodes can do repairs on their behalf as well.

During the regeneration of the parities the node that performs the repairs uses pp-tuples (two p-blocks) and dp-tuples (one d-block and one p-block). Figure 7.1 indicates the tuples that are used at *node 1*. Table 7.1 enumerates the operations involved during the regeneration of the parities located in faulty nodes 2, 6 and 11. To illustrate with an example, it presents the steps done while repairing the parity block $p_{21,26}$. Steps 1-3 and 5 are simple local operations. Step 1. *Node 1* computes the dp-tuples *id* associated to the missing parity. The *id* is a unique identification associated to each lattice's element, for example, it could be a hash. Step 2. *Node 1* chooses one tuple, e.g., $\{key_{21}, key_{16,21}\}$, according to its availability status or other node's constrains. Step 3. *Node 1* computes the location for p-block $p_{16,21}$ (*node 5*), d-block d_{21} is stored locally. Step 4. *Node 1* gets the parity from *node 5*. Step 5. Finally, *Node 1* recovers the p-block by computing $p_{16,21} \oplus d_{21}$. The recovered p-block should be restored to the network by uploading it to another node. Similar steps are done to recover other missing elements.

7.2 Use Case: Disk Arrays

Disk arrays can increase the performance of a system because individual requests are served faster when multiple disks collaborate to increase throughput. In addition, the possibility to answer multiple requests in parallel gives the chance to increase the system's I/Os rate. Some form of redundancy is required to compensate the increase of combined disk failure rate when writing to disk arrays. RAID organizations (Patterson et al., 1988) are a well-established solution for the industry and the home user to increase the performance and reliability of large arrays of inexpensive disks but the exabyte scale redefines storage needs. Although there is still future for cheap spinning disks in data centers and home solutions, the market changed substantially in the last decades. Some of the problems that face the community are:

1. *Rebuild time* may take hours or days. Systems that tolerate 2 failures dominate the market. More failures during rebuilds are a source for data loss.

2. The *disparity between technology development* means that disk capacity has grown

steadily while bandwidth lays behind.

3. *Research on RAID lost popularity* in favor of cloud-based storage solutions, even though RAID are still in use for many reasons, e.g., privacy. Google Trends (Choi and Varian, 2012) reveal that the popularity of "cloud storage" equalled "RAID disks" in Aug 2009. Since then, searches for cloud storage continues to increase.

4. *Failure rate* is mostly related to system's size (Schroeder and Gibson, 2010).

5. *Failure correlation* is frequently disregarded. The assumption that failures are independent and that time between failures is exponentially distributed is not valid (Schroeder and Gibson, 2007). Failure correlation should be considered.

RAID stands for redundant arrays of independent[1] disks. We propose to rethink disk arrays with the use of alpha entanglement codes. In this context, it make sense to drop the term "independent" and replace it with "interdependent". Note that this use case is valid for log-structured append-only storage systems.

7.2.1 Entangled Mirror

In Chapter 3, we discussed two array organizations based on simple entanglement. Simple entanglements require equal numbers of data and parity drives; therefore, the array will have the same space overhead as mirroring. A quick recap of that discussion will provide completeness to this subsection.

Full partition. In this approach, blocks are written sequentially on the same drive. The process does not spread content across drives. Most of the drives will remain idle and can be powered off as in a massive arrays of idle disks, MAID, configuration (Colarelli and Grunwald, 2002), which could result in significant energy savings.

Block-level striping. This second approach distributes data over all available drives to improve performance.

In an entanglement chain, blocks that are located at the extremities have less redundancy. This problem has more impact on full partition where the content that is at the extremity of the chain is equivalent to all the data stored in one disk. At block-level striping, the amount of data at the extremity of the chain is equivalent to the block size. The same chapter presented open and close entanglements, where the second approach addresses this problem. We showed that in full partitions both solutions provide better five-year reliability than mirroring, reducing the probability of data loss by respectively 90 and 98 percent.

[1]Patterson's paper used the word "inexpensive" but since hardware controllers became highly sophisticated the term independent was gradually adopted.

7.2.2 RAID-AE

One misconception often held is that storing data in disk arrays or do-it-yourself backup is an outdated solution. The growing cloud storage market and the scalability issues of RAID make organizations consider moving data to the cloud. However, in-house storage is still used by many industries, e.g., finance market, government, and research institutions. On the other side, many cloud storage services are built on infrastructures that use RAID-like storage solutions. For example, Backblaze's online backup service uses a RAID system built on top of Linux and Reed-Solomon codes (Beach, 2015).

There are three ways to increase the redundancy of data when configuring disk arrays: mirroring (discussed previously), parities and erasure codes. Reed-Solomon codes became a sort of de-facto industry standard for erasure coding, and particularly to archive data. Perhaps the main reason for their broadly acceptance is because, when the storage market became to grow, they were already an established solution for digital communications. In brief, Reed-Solomon codes are storage efficient, well-understood, and open-source libraries are available. The main problem is the bandwidth and I/O cost during the repair process. An attractive solution commonly seen in RAID-like systems are constructions based on combinations of Reed-Solomon blocks that are optimal locally repairable (Papailiopoulos and Dimakis, 2014).

Scalability. RAID-AE can make improvements for both horizontal scalability and vertical scalability. In a RAID-AE array, it is possible to add more disk units to convert, for example, a 10-disk array into a 11-disk array or add more capacity to each of the disks in a 10-disk array. Depending on the implementation, both actions can be done dynamically without interrupting the array and without encoding data again.

Never-ending stripe. Since entanglement chains can always grow by creating a never-ending stripe, they change the classical notion of *stripe*. For clear examples, we use RAID5 but with more complex RAID organizations the situation remains the same. The way to compute parities is based on a fixed-width stripe, e.g., in a 6+1 disk RAID5, one parity unit is computed using 6 data units. When one disk is added to the array, the new array 7+1 disk RAID5 requires re-encoding parities, one parity unit is now computed using 7 data units. Additionally, there is a large penalty when a disk fails because of the bandwidth and I/O overhead to repair each missing unit. Parity declustering (Holland and Gibson, 1992) reduces the rebuild time by distributing repairs across devices in a large array but this technique cannot reduce the overhead mentioned above. We argue that the scope for major improvements is limited by the stripe size. Increasing the length of a stripe has a prohibitive effect in repairing data.

Degraded reads. The stripe size impacts on degraded reads in RAID-like solutions. In data center environments where disks are distributed in different machines and racks, the performance of degraded reads matters. It is desirable to overcome temporarily unavailable data, in many cases, single-failures due to software updates, schedule restarts, etc.

Other features. RAID-AE can be implemented to provide distributed repairs and load balance for read intensive workloads. One important difference in our model is that we do not assume failure independence and because the parameter α can be changed in future, the system can scale in fault-tolerance. In sum, RAID-AE will permit different arrangements with trade-offs between capacity overhead, network bandwidth overhead, rebuild time and fault tolerance.

7.3 Discussion of Strategies for Realistic Implementations

Our long term goal is to implement entanglement codes in real-world storage applications. This section aims to discuss further aspects of the codes that can provide the grounds for future implementations of an entangled storage system. Alpha entanglement codes are not restricted to a particular storage architecture. They could be implemented in a DHT-based peer-to-peer collaborative storage system, in redundant disk arrays, in the cloud, or even in a cloud of clouds. Discussing all the complex characteristics of each implementation is far beyond the scope of this thesis but we address some questions that have come up during our investigations.

The implementation of a prototype built on top of a distributed hash table (DHT) with similarities to the use case presented in Section 7.1 was a practical way of identifying technical challenges related to integrating entanglement codes with an existing system. The rationale for using a DHT is that data blocks can be placed on geographically distributed nodes, possibly under distinct administrative domains, which fail independently and hence provide good tolerance against faults and attacks. Some of the topics discussed in this section come from the experience gained in building this prototype and other entangled based solutions.

7.3.1 Integrating with an Existing System vs Building from Scratch

This section presents a brief account of our attempts to implement entangled based solutions.

Cassandra-HEC

Cassandra-HEC is a prototype built atop Apache Cassandra (Lakshman and Malik, 2010). Cassandra is a data store written in Java that uses a hybrid model between key-value and column-oriented databases. It was initially developed by Facebook and subsequently released as open source. Nodes are organized in a cluster/ring, where each node plays the same role so that there is no single point of failure. Our deployment is a two-tier architecture: the lower tier, composed of Cassandra storage nodes and the upper tier consists of broker nodes that entangle and disentangle data to provide high data reliability. Brokers can reside in the same network of Cassandra nodes to reduce the bandwidth cost, and a single node can act as both a broker and storage server. Cassandra-HEC takes its name after helical entanglement codes. The system relies on entanglement codes to store data redundantly instead of Cassandra built-in replication. Cassandra replication factor is set to 1, which means there is only one copy of each row across the cluster.

The major challenge was to implement our own placement policies. Cassandra uses a partitioner to determine how data is distributed across the nodes in the cluster. The partitioner has a key role in Cassandra's architecture to assure scalability. The application supports three partitioners: Murmur3Partitioner, RandomPartitioner and ByteOrdereredPartitioner, however, the last two are only provided for backwards compatibility. By default, data is distributed across nodes using Murmur3Partitioner, a consistent hashing strategy, whereby keys are hashed to determine the node responsible for storing the associated value.

From our discussion with Cassandra experts, we found that non-default partitioners are highly discouraged in order to maintain load balance. The problem with this policy is that we found that some applications including ours need more control on placements and, at least in our case, we can handle load balance in a different way. Our aim was to test our block allocation policy that map blocks deterministically and evenly to different failure domains. Eventually, we implemented it using the ByteOrdereredPartitioner, but we do not know if the support to this policy will continue in future versions.

Ceph

Ceph is a storage system designed for scalability, performance and reliability. It is the outcome of research done at the Storage Systems Research Center[2], University of California, Santa Cruz. Recently the project was acquired by Red Hat, who continues its development as an open-source project. It has support for replication, and to local repairable codes based on Reed-Solomon. It is notably used by CERN, Deutsche Telekom, Cisco and various universities.

We discussed with Ceph engineers about the possibility of implementing helical entanglements. Again, the main challenge was the lack of support for *flexible placements*, where flexible placements means a fine-grained policy enforcement. At that time, there were plans to provide more flexibility but we are not sure how those plans developed. Giving more flexibility to the customer increases the risk that a bad configuration can cause problems, e.g., like the load balance issue mentioned by Cassandra engineers. However, discussions with Ceph engineers suggested that they are open to support more flexible schemes. In fact, entanglement codes were considered as a good example of the kind of applications Ceph should support in a flexible placement policy[3].

Ceph maps objects to placement groups (PGs) so the group is mapped to object storage devices (OSDs). The main purpose of the PGs is to reduce the amount of metadata per-object. Additionally PGs improve load balance, for that reason at least 100 PGs are recommendable. Our concern is to mapped objects to a large number of OSDs. The idea of data entanglement

[2]http://www.ssrc.ucsc.edu, Mar 2016

[3]Discussions between Ceph developers about helical entanglement codes and flexible placements: https://web.archive.org/web/20170327085721/http://www.spinics.net/lists/ceph-devel/msg29903.html

is that all blocks in the system are somehow connected to each other. Therefore, blocks must be spread evenly in a large number of OSDs. Ceph was initially designed for mirroring replicas. Recently the system incorporated support for Reed-Solomon and local repairable codes but none of these codes require access to a large number of OSDs. In fact, these codes require to distribute a group of only 10-20 blocks in distinct failure domains.

HEC CoC

A cloud of clouds (Slamanig and Hanser, 2012; Bessani et al., 2013) is an architecture that combines multiple cloud vendors for achieving at least one of these goals: avoid the lock-in problem, increase fault-tolerance and availability, increase bandwidth and privacy. The *lock-in problem* (Armbrust et al., 2009) occurs when the client founds prohibitively expensive to change providers, an action that requires migrating all data from one cloud to another. Bandwidth charges is one of the strategies to refrain customers to jump into a cheaper service. Even if not common, cloud providers experience failures that affect their services badly, hence, dispersing data across multiple providers increases fault-tolerance and availability. Providers that give unlimited cloud storage usually put limits on the upload/download bandwidth. Finally, privacy is a big concern for cloud adoption in organizations and home users (Zhou et al., 2010).

HEC CoC is a proof of concept for a cloud of clouds based on entanglement codes. A small demo was presented during our annual Sinergia project meeting in March 2015 at ETH Zurich. This application was built from scratch, without any knowledge of the placement policies enforced by the selected cloud providers. Data is encoded with helical entanglements and distributed with a coarse grain placement policy to three different cloud storage services: Google Drive[4], Microsoft OneDrive[5], and Dropbox[6]. At the time of the demo, Dropbox was outsourcing the storage to the Amazon cloud.

We used graph colouring to distribute blocks across the three providers. It is a coarse grain placement strategy because remote placements are not know. However, it is safe to assume that the blocks that are assigned to the same provider are stored with additional redundancy since providers have their own reliable mechanism. On our side, the graph colouring assured that if one provider is not available data can be repaired using blocks stored in the other two destinations. Additionally, HEC-CoC can be used for privacy if the graph coloring is done with the constraint that each provider does not receive enough information to recover data. The use of entanglement codes as an information dispersal algorithm opens new research opportunities. Other researchers explored data dispersal algorithms to create cloud of clouds storage networks (Li et al., 2014).

[4]https://www.google.com/drive/, Mar 2016
[5]https://onedrive.live.com/about/en-us, Mar 2016
[6]https://www.dropbox.com, Mar 2016

7.3.2 What to Entangle

The first question that arises while implementing the encoder is which blocks are going to be entangled together, i.e., located in the near vicinity of the lattice. The entanglement encoder is fed with blocks but the algorithm does not know anything about them. *What is the data source of the blocks? Is the system mixing blocks of a large unique file? Is the system mixing blocks from files of the same user or from different users? Do the blocks have equal size?*

The answers depend on many factors such the purpose of the application, the workload, resource contention, etc. In the particular case of Cassandra-HEC, the user stores the files in a special backup directory. This directory is frequently scanned by a service that reads the files and prepares them for the entanglement. Files are striped and blocks are accumulated in a storage pool. The encoder takes elements at random to do the entanglement. Blocks have identical size (1MB). It is expected that more variables in the system like coexistent variant-size blocks translate in additional metadata. But we can predict that the regular behaviour of the entanglement algorithm reduces the amount of effort needed to handle size-variant blocks from diverse sources.

7.3.3 Where to Entangle

There is not a single answer regarding the location of the encoder. We suggest possible options by considering two more use cases:

- Use case 1: In a geo-replicated backup data is encoded at the client.
- Use case 2: In a RAID-AE entanglements could be performed by a hardware RAID controller using the encoder presented in Section 4.8.
- Use case 3: In a confederate geo-replicated backup, data is encoded by middleware nodes. This entanglement model will allow the entanglement of files from various users.
- Use case 4: In a highly centralized datacenter, data is encoded at the backend servers. Clients transmit plain (or encrypted) but not encoded data to the servers.

7.3.4 Where to Store Blocks

As we previously mentioned, block placement policies are an important aspect in the design of an entangled storage system. Previous work on placements are based on traditional reliability methods in which the size of redundant groups is small. For example, replication requires the distribution of k copies among distinct storage devices, where k is typically two or three. On the other hand, systems that use traditional erasure codes like Reed-Solomon split the data fragment into k fragments and generate m encoded blocks. As a result, only $k + m$ blocks must be distributed to different devices, with $k + m$ being in the order of 20 devices.

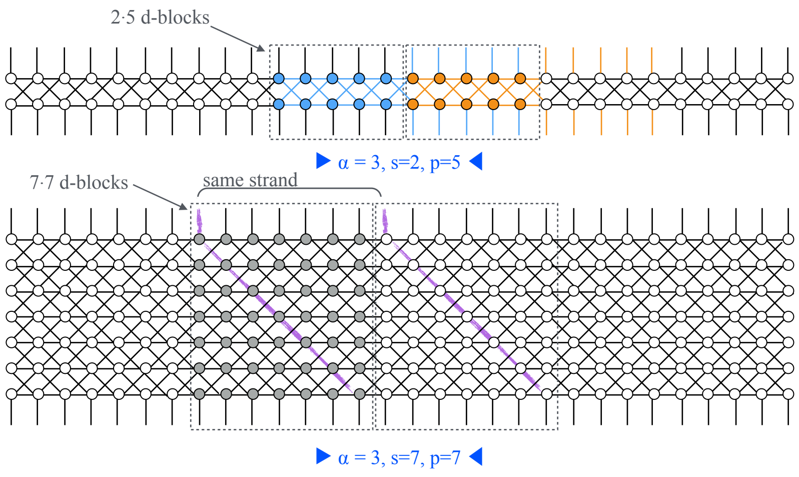


Figure 7.2 – Lead windows for AE(3,2,5), i.e., 5-HEC, and for AE(3,7,7).

Entanglement codes behave substantially different to conventional approaches since all blocks in the system are somehow connected to each other. A straightforward practice is to store all redundant data in different failure domains. Yet, the total number of storage devices in any realistic installation is limited. Nevertheless, the number of blocks that should be stored in different devices can be restricted to the size of two adjacent *lead windows*. A lead window is defined by the code parameters s and p , i.e., it is the interval during which one helical strand revolves around an imaginary axis. This interval is part of the extended near vicinity for one d-block. Figure 7.2 shows lead windows for 5-HEC and other setting.

The repair process works iteratively using elements of the near vicinity of the missing element and expands the region when there are not enough blocks for repair success. The near vicinity of border d-blocks falls in two lead windows. The repair process handles the lead window as a sliding window. Therefore, the elements (d-blocks and p-blocks) that are located in a lattice region defined by two adjacent lead windows should be stored in different devices. The number of elements in a lead window is computed from the code parameters as in

$$(\alpha+1)\cdot s\cdot p, \tag{7.1}$$

since one window encloses sp d-blocks and for each d-block there are α p-blocks. For instance, a 5-HEC lattice requires at least 80 $(2\cdot4\cdot2\cdot5)$ devices to assure that the actual size of an erasure pattern does not become smaller than its theoretical size. To wrap up, if the effective size of a pattern becomes smaller, the effects of redundancy propagation are reduced. Moreover, smaller erasure patterns are more likely to appear. For that reason, it is undesirable that neighbor elements are stored in the same device (failure domain).

We consider three different allocation methods:

Round Robin Placement. This policy uniformly distributes blocks in distinct disks of an array

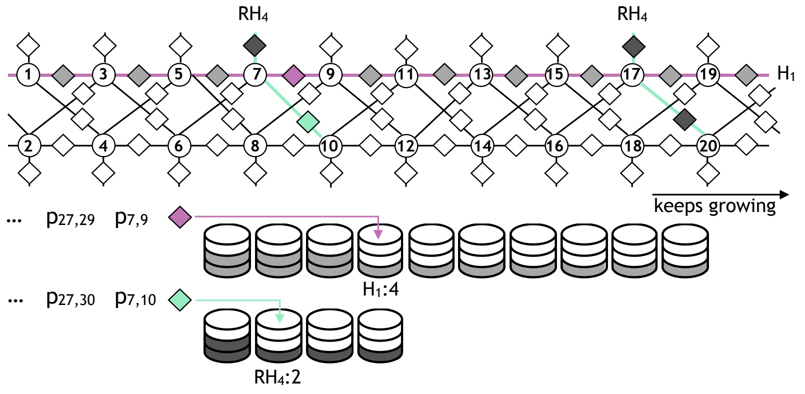


Figure 7.3 – Round robin placements for AE(3,2,5) using a disk array per each distinct strand. The image shows the disk array for right-handed strand RH_4 and horizontal strand H_1. Parity blocks are explicitly shown for the sake of clarity. As the lattice grows, blocks that are at a constant distance are stored in the same storage device. For example, $p_{7,9}$ is stored in the fourth disk of H_1 array. The next p-block $p_{9,11}$ is stored in the fifth disk. By writing p-blocks in circular order, the next p-block to be written in the fourth disk is $p_{27,29}$, which is at $2 \cdot p$ d-blocks distance on the horizontal strand.

using circular order. Figure 7.3 shows AE(3,2,5) with round robin policy in which each strand has its own array. For example, a 10-disk array stores the parities of the H_1 strand. Disk four ($H_1 : 4$) stores the $p_{7,9}$ and any parity located at a multiple of 10 hops on the same strand, e.g., $p_{27,29}$. The minimum length of the array is determined by the number of elements in a certain strand in a two lead window interval. This requirement assures that all elements that are at a short distance will be stored in different failure domains. The figure also shows a 4-disk array that stores the parities of the RH_4 strand. The RH_4 array is shorter than the H_1 array since there are only two p-blocks inside a lead window that belong to RH_4, while there are five p-blocks that belong to H_1. The setting $AE(3,2,5)$, according to Equation 4.1, requires twelve arrays to store the parities of the twelve distinct strands and one array to store the d-blocks, in total 80 disks.

It is worth mentioning that each strand class (H, RH, or LH) contains all the necessary information to regenerate the storage content. That means that all the arrays associated to the same strand class, e.g., the five RH_x arrays in our previous example, could be placed in one geographical location and the arrays associated to a distinct strand class placed in another location to tolerated network partitions.

The Cassandra-HEC prototype was implemented using round robin placements, assigning blocks to a smaller set of nodes as in a RAID-like system. As previously mentioned, a problem with this placement approach is that implementation might be difficult especially if entanglement codes have to be integrated with an existing system.

As we became interested in the implementation of entanglement codes in peer-to-peer networks, a simple but realistic way of mapping data blocks to storage devices is a random placement policy.

Random Placement. This policy is perhaps the most common way of distributing blocks among a set of devices. This approach has been proposed and studied in the theory (Karger et al., 1997) and many systems use a consistent hashing variant: Chord (Stoica et al., 2001), Sorrento (Tang et al., 2004), Crush (Weil et al., 2006b), Dynamo (DeCandia et al., 2007), Cassandra (Lakshman and Malik, 2010). Spreading blocks evenly over nodes is a well-known scalability problem that affects load balance and system's reliability. The search for a scalable solution has been studied in three main directions: table-based, rule-based and pseudorandom-based distribution. Prior work from other authors studied the advantages and disadvantages of each strategy (Miranda et al., 2014).

We use random placements for the simulations used to evaluate complex entanglements and spigot codes. Those studies demonstrated that entanglements can be used with random placements and still achieve high reliability. And it opened the possibility of using entanglements in peer-to-peer networks. An interesting follow up study would be to look into the problem of replica placement in dynamic environments.

Copyset replication (Cidon et al., 2013) is an alternative approach to random placements. As explained by the authors, combinatorics has an important role in the probability of data loss. They gave the example of a system with 9 nodes and a replication factor of 3, in which nodes are split in three copyset groups: $\{1,2,3\}$, $\{4,5,6\}$, and $\{7,8,9\}$. Such systems can experience data loss if the three nodes that are part of the same copyset group fail simultaneously. On the contrary, in random placements and with sufficient blocks stored in the system, any combination of three nodes failing simultaneously can cause data loss. Obviously, the same principle applies to our round robin placements. Distributing data in the same disk arrays reduces the number of existing fatal combinations of nodes that will exist in random placements. However, this reasoning is not conclusive as the total number of blocks and total number of storage devices are to be included in the probabilistic model.

7.3.5 How to Reclaim Storage

On one side, entanglement codes are designed for permanent storage. Therefore, the lattice design is not adequate for deleting random data. On the other side, the lattice grows regularly with new blocks added to the head of the structure; the tail contains the oldest data. This design is optimal for implementing an entangled storage system that reclaims space by deleting aged data, as it commonly occurs in a log-structured file system (Rosenblum and Ousterhout, 1992).

Reclaiming space requires the definition of the *retention duration* policy, which is the minimum amount of time that the data is preserved in the system. This policy could be associated to files, to a group of files or to files belonging to a certain user. What follows is related to a question we answered before: "what to entangle". The retention duration must be specified before data ingestion to allow colocating blocks with equal retention policy on the same entan-

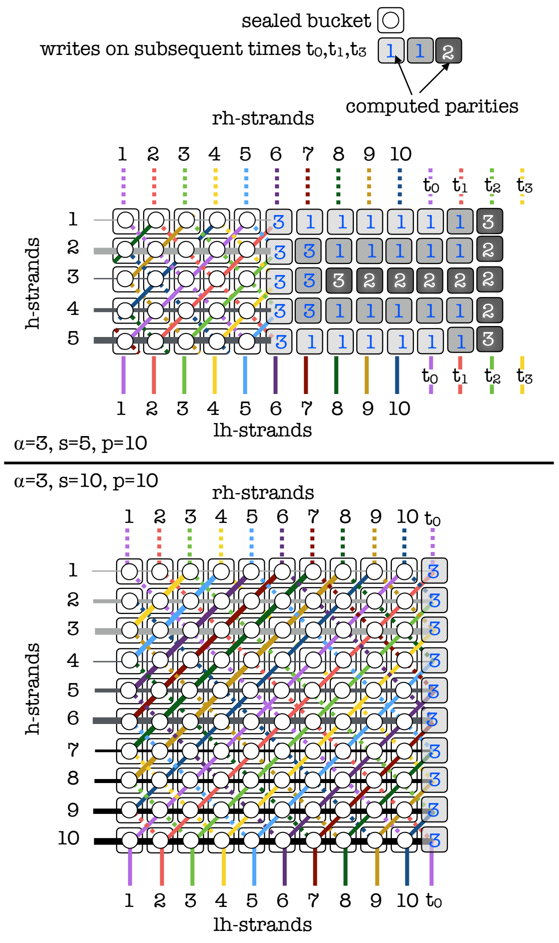


Figure 7.4 – Write performance for two code settings. When $s = p$ full-writes are optimized since all the parities needed to complete triple-entanglements and seal the buckets are available in memory. When $p > s$, one option is to do full-writes for s elements, a second option is to write buckets partially with the parities available in memory.

gled structure. In this way, the system is composed of multiple lattice structures with different retention policies and fault tolerance strategies (each lattice could be built with different code settings). A retention period could be extended by reinserting data at the head of the lattice in case it needs to be preserved for a longer time than anticipated.

Cassandra-HEC is a rolling storage database. Its built-in feature *time-to-live attribute* is useful to claim space. Each block is stored for a certain amount of time or until the database reaches a certain level of its capacity. The maximum lifetime of blocks in our implementation is capped at 20 years by Cassandra's time-to-live (TTL) attribute.

7.3.6 Which Code Parameters

In RAID terminology, a *full write* or *full-stripe write* occurs when the full stripe is written at once. This operation optimizes the writing overhead that occurs in parity update operations. Even though entanglement codes do not have such notion for stripes, write optimizations exist. The values of code parameters s and p impact the writing performance. When $s = p$, the number of data blocks that can be fully entangled in parallel operations is maximized.

We define a *bucket* as a container for a data block and its α output parities. A bucket also receives α input parities that are used by the encoder to compute the outputs. A *sealed bucket* is a bucket where the α parities are already computed, see Figure 7.4. In other words, a sealed bucket contains a fully entangled data block. This is possible only when the α input parities needed in the process are already computed and kept in memory. The memory requirement for full-writes is $O(N)$, where N is the number of parities computed in the full-write. The illustration shows that when $s < p$, many buckets have partial inputs, allowing us to compute some parities but not to seal the bucket.

7.4 Summary

This chapter discussed technical challenges in the design of an entangled storage system. In particular, it provided useful insights for the block placement problem, shared lessons learned during the prototyping of entanglement codes and gave recommendations for future research directions. It considered various environments in which entanglement codes could be used: a centralized storage solution in a datacenter, a cloud-of-clouds backup solution, an array of disks, and a geo-replicated backup service. It raised six questions that may appear in the design of an entangled storage system and provided possible answers considering a variety of conditions.

8 Conclusion

I say that the Library is unending.
The idealists argue that the hexagonal rooms are a necessary form of absolute space,
or at least, of our intuition of space [...]
Let it suffice now for me to repeat the classic dictum: The Library is a sphere whose exact
center is any one of its hexagons and whose circumference is inaccessible.
— The Library of Babel, Jorge Luis Borges (1899-1986),
Argentine writer.

As systems scale in size and complexity, it seems wise to review what we know. By questioning the conventional methods to create redundant information we discover new things and learn how to harness redundancy. The concept of redundancy is powerful, [1] and survivability comes through the strength built with redundancy. Storage systems store data redundantly to ensure that the content will survive failures. To answer how long it will survive we would have to study its reliability, but reliability measurements are sometimes too optimistic. Nevertheless, many systems are built with two failures fault tolerance based on the assumption that the probability of a third or fourth failure is extremely low. A low level of fault tolerance ignores many issues such as high availability in wide area networks, correlated failures like power outages, datacenter failures and data corruption. The pitfalls of the current redundancy paradigm is that fault tolerance increases linearly with storage in the case of replication and proportionally with bandwidth and input/outputs operations with classic codes.

A dependable system is trustworthy to its users. It is a challenge to build redundancy methods that fully leverage system resources and consider all aspects of dependability: availability, reliability, safety, integrity and maintenability. To compensate, systems grow in complexity trying to find good compromises, some problems are ignored, and the costs are rarely discussed. We

[1] The original phrase: "This thing called redundancy is a powerfool tool" belongs to Paul Baran.

propose entanglement codes to have more options when trying to address these limitations. This new alternative creates an unending layer of interdependent resources. This layer can coexist with other layers, each with its own security requirements (code settings), to protect content in a potentially ab aeterno storage system.

This dissertation deals with the design of practical erasure codes for storage systems. The research challenges related to coding require interdisciplinary studies that bridge the different expertise to design practical solutions under realistic assumptions. Chapter 2 examines extremely different topics to find unapparent connections in an attempt to close the gap between the information theory, storage and cryptography communities. Thus, this chapter contributes with an extensive literature review. The survey is organized in four main sections: information theory background, codes for storage applications, the storage sector, and data entanglements. This holistic approach prepares us to understand coding methods from different angles and find sinergia between existing codes. Particular attention is paid to the entanglement of user files, proposed by other authors to protect files against censorship. Used as a censorship deterrent mechanism, the rationale was to make deletions expensive. And that was the original goal of our study too. But when the connection between entanglements as censorship resistant mechanisms and entanglements as redundancy mechanisms for dependable systems became more prominent, we set our goal for the second option. The connection resides in the fact that the anti-censorship method goal is to cascade deletions, i.e., destroying a file destroys other files as well, whereas the dependable method's goal is to recover data from whatever remains available in the system. In both cases, there is a set of documents (or data blocks) that have the all-or-nothing integrity, either all elements survive or none of them does. Entanglement models became worthy of study with the aim to leverage interdependencies between content and propose practical erasure coding solutions. From Chapter 3 through the end of the dissertation we reduce the complex space to the realm of storage systems, in particular to the study of entanglement codes that propagate redundancy across storage locations. Simple entanglements, complex entanglements and beyond entanglements address the questions of how to connect data blocks and leverage this connections to build a highly fault tolerant, long-term storage solution in a cost-efficient way.

Open and closed entanglements are uncomplicated entanglement codes. In spite of its relative simplicity, the model is powerful. The idea behind simple data entanglement is to intertwine data and parity blocks using the bitwise XOR operation with the goal of *refreshing* previously calculated parities and increasing the scope of redundant information. The process builds a chain that associates old information to the new data. In this way, old blocks will become indirectly dependent on blocks that will be inserted in future without the need to run the algorithm more than once. Simple entanglements require equal numbers of data and parity drives; therefore, they have the same space overhead as mirroring. We have studied open and closed entanglements, two simple data distribution layouts for log structured append-only storage systems. While open entanglements maintain an open chain of data and parity drives, closed entanglements include the exclusive-or of the contents of their first and last data drives. We have described two possible array organizations: full-partition write and block-level striping. The full-partition approach writes blocks sequentially on the same drive. This data layout is best suited for archival applications with very low read to write ratio. The block-level striping distributes data over all available drives to improve performance. We have used Markov models to evaluate the five year reliabilities of open and closed entanglements for two different array sizes and drive failure rates and argued that open and closed entanglements provide better reliability than mirroring.

Complex entanglements combine simple entanglements to build a helical lattice. This study leads to the definition of helical entanglement codes and its generalized model alpha entanglement codes. At the core of this complex mechanism there are simple rules to propagate redundant information across a large scale distributed system. The rules create a virtual storage layer of highly interconnected elements. We have studied the fault tolerance, evaluated the impact of entangled data in different simulated environments and argued that entanglement codes offer high fault tolerance due to the redundancy propagation controlled by three parameters. The parameter α classifies the code families in single-entanglements (equivalent to simple entanglements), double-entanglements and so on. The possible repair combinations grow exponentially with the value of α. At the same time the storage overhead of the code grows linearly with the same parameter, e.g., double-entanglements are equivalent to three-way replication and triple-entanglements are equivalent to four-way replication. The other two parameters increase fault tolerance even further by controlling the vicinity of the blocks without the need of additional storage or without changing the single failure repair cost. We have conducted a theoretical analysis of fault tolerance and reliability of entanglements. We compared them with classical codes and argued that alpha entanglements are practical codes that excel at code locality, hence, they reduce repair costs and become less dependent on storage locations with poor availability. The interdependencies between content offer additional data integrity as they make more difficult to modify data undetectably. We have run experiments to answer how much redundancy is needed to keep data safe in unreliable environments without requiring maintenance and showed that entanglement codes outperform classical codes in many disaster recovery scenarios.

Our final contribution addressess a limitation of entanglement codes. Classical codes can increase storage overhead in a gradual manner controlled by their settings, e.g., a RS(6,2) increases storage with a factor of about 1.33x. However, the only parameter that controls storage overhead in entanglements is α, which can only increase at integer steps. For example, single-entanglements add one encoded block per data block, double-entanglements add two encode blocks; in other words, the storage overhead increases in steps of 100%. This "entry space requirement" is expensive for some applications. We have investigated the suitability of different techniques to increase their code rate and reduce the space footprint from entangled data. Our new solution, spigot codes, present four types of code puncturing and XOR schemes built on top of entanglement codes. The studied examples reduce up to 50% of the storage requirements of vanilla entanglement codes without significant loss of entanglement's properties. The number of repair rounds in a spigot code does not increase significantly unless a disaster affects many locations (in the order of 30% or, in some cases, 40% of the storage locations). The simulations provide evidence in support of the flexibility of entanglement codes and their derived codes to cover different needs.

8.1 Research Limitations and Future Directions

This section reports the limitations of this study and suggests possible directions.

8.1.1 Beyond $\alpha = 3$

In this study we do not cover entanglement codes with $\alpha > 3$. We can safely speculate that the fault-tolerance would improve substantially. We are confident that the upper bounds for fault tolerance are defined by the size of an α-hypercube. This upper bound impacts on the size of minimal erasure patterns $|\mathrm{ME}(x)|$ with $x = 2^\alpha$, for example, for $x = 16$ redundancy propagation would be enclosed in a tesseract (4-cube) composed of 16 nodes (d-blocks) and 32 edges (p-blocks). In addition, for larger α values we expect improvements in repair performance. However, it is not clear how to connect the extra helical strands. In double- and triple-entanglements the rh- and lh-strands are defined along a diagonal of slope 1. One possible option is to add additional helical strands with a different slope.

8.1.2 Security Assessment

The simulation framework used a random distribution of failures. The study covers a sheer number of arbitrary failure patterns that are part of large failure scenarios in which up to 50% of the storage content is not available. However, random failures do not give full information about the survivability of entangled data under certain attack models. This raises the question if entanglement codes provide a strong mechanism against censorship.

Entanglements as a censorship deterrent mechanism are effective if destroying a document is expensive to the attacker. A priori, we know that the upper bounds of propagation might become the Achilles heel for a powerful adversary that tries to destroy particular files. We have seen that in double-entanglements the upper bound is a square. It is trivial to verify that it is composed by 4 d-blocks and 4 p-blocks. It is plausible that entanglement codes have limitations to deal with malicious attacks. One could argue that destroying 7 additional blocks to target 1 block does not seem very expensive. But such argument suffers from a number of pitfalls. To begin with, it assumes that the censor knows which blocks are neighbors. Entanglements can be done by the client, in a middlware layer or at the storage backend. Moreover, they can be done intra-file or inter-file, from the same user or from multiple users. In this way, the implementation can change substantially the attacker's options. As an additional remark, we cannot ignore that security measurements make the system costlier. Entanglement codes are designed as a practical solution, therefore the main goal is to protect data against the most common failures. Finally, the upper bounds of propagation are a cube in triple-entanglements and a tesseract for quadruple-entanglements. By increasing the number of elements exponentially and reducing the storage overhead with spigot codes built on top of the original method, entangled data may have enough protection against malicious attacks. This research direction seems to be promising.

Finally, a malicious attacker may try to tamper data. Notwithstanding that evaluating the robustness of an entangled storage against malicious attacks was not a primary goal in this research, data integrity is nonetheless an emergent property of entangled data. Intuitively, if an attacker wants to tamper with a certain data block, he has to manipulate at least all p-blocks that were computed in the α chains to which the block belongs. Clearly, the age of the data becomes a natural deterrent mechanism. We cannot rule out other more complex mechanisms of data manipulation that may appear in a particular implementation.

8.1.3 Reliability Assessment

We did not conduct a Markov model analysis for complex entanglements. Doing a detailed analysis of failure patterns, as we did for simple entanglements, is extremely hard. As an alternative, we have evaluated reliability based on combinatorial methods. In particular, we have estimated the probability that the system will successfully answer a block read request. An entangled system has a large number of possible paths to reply to the same request. To reduce cumbersome calculations caused by all existing paths we have pruned many of them. Consequently, our results undersell entanglement codes, especially when the result is compared with classic erasure codes that consider all existing paths (since their computation is well known). Finally, an additional limitation of this study is that the final result was given for a certain code setting. It is not easy to generalize the formula to other settings and all the computations should be done again for other settings. However, the obtained formula is useful to initially assess the code and compare it against its alternatives.

8.1.4 Block Placements

We have developed different tools that allow testing connections, verifying failure patterns and characterizing code performance. This environment is an ideal research playground for examining entanglement codes and other erasure codes in order to run comparisons and choose the best coding parameters. This simulation framework is designed to study what happens with data in environments that either accumulate plenty of failures before repairs take place or large number of failures happen all at once. A caveat is that when distributing redundant data in a large scale storage system, the number of locations and the way data is distributed plays an important role in the chances of data to survive failures. Spreading blocks evenly over storage locations is a well-known scalability problem that affects load balance and system's reliability.

In the experiments that consider placements, we run simulations with 100 distinct locations.

Since we used random placements, it is possible that blocks that are needed in the repair of other blocks are all or some assigned to the same location. When this situation happens, the system becomes less resilient to failures. As expected we have found that some redundancy in classic codes is degraded due to this problem. The analysis was documented in the previous chapters. We believe that our findings compare well with other solutions that use random placements, however, we should sound a note of caution with regard to the results of classic coding. The number of distinct locations that we used is not enough to support a fair distribution and we did not do anything to restore balance before running our experiments. When running experiments with 1000 distinct locations, most of the blocks where fairly distributed. Despite the lack of perfect balance caused by fewer locations, no significant differences were found with 1000 locations.

For the case of entanglement codes, running experiments with 100 locations seemed a priori to violate redundancy even more than in classic codes. For instance, RS(10,4) requires 14 distinct locations for a fair distribution, whereas AE(3,2,5) requires at least 80 distinct locations. Since entanglement codes have a radically different repair model, their can take advantange of a large number of locations. Initially we thought that the unbalance caused by a restricted number of locations would be counterproductive for entanglements. However, a closer inspection on single failure repairs revealed that entanglement codes are able to take advantage simultaneously from local, zonal and global repairs.

Further evaluations on the impact of placement rules in different system sizes are needed. We are developing a playground to simulate placements and other relevant factors that have an impact on a storage system. This work-in-progress may become a useful resource for researchers and engineers working on the design of reliable storage systems.

8.1.5 Entanglements for Communications

The study of entanglement codes was focused on addressing problems for storage systems. Perhaps, the schematic encoder design given in Chapter 4 was the closest we got to a solution that can be used for data communication. We provided tables that show how the code works even at bit level. As the focus of the study was on storage solutions, we were unable to investigate how to decode entanglement codes at the bit level without knowing the position of the error to use them as error correcting codes. Further studies are needed to find if entanglements are useful codes for error correction in data transmission applications. In particular, it may be useful to determine if probabilistic decoding gives good results in entangled data.

8.1.6 Performance Measurements

Our research failed to give results on coding/decoding speed. In our first set of experiments the simulations performed encodings and decodings but did not consider storage locations. Therefore all the process was performed locally without taking into consideration blocks distributed in geographically distant locations. Most of the evaluations are based on simulations that do not perform any actual encoding. This method was chosen because it is more rapid and encoding is not necessary to study fault tolerance, placement policies and the performance of the code when solving single-failures.

Although our simulations do not capture actual coding/decoding speed, we know that entanglement codes have a low coding and decoding complexity and they are based on lightweight XOR operations. It can thus be reasonably assumed that it is possible to implement very efficient encoder/decoder implementations. This assumption appears to be supported by a short demo of the prototype HEC-CoC given in our annual research project meetings. There, we showed that files could be encoded and decoded in a reasonable time in spite that blocks were stored and retrieved from multiple clouds.

8.2 Foresight and Hindsight

The task that initiated this study was to mix data blocks in order to improve the safety of a trustworthy storage system. During the time I was struggling to find efficient ways of connecting data blocks I had no initial constrains that could bias my decisions. Except, for three conversations that resonate with me and influenced the design of entanglement codes.

The first conversation was with Prof. G. Venturino. It occurred when he taught me circuit analysis many years ago. To gave me a lesson, he asked me to solve the well-defined nine dots problem. The goal of this game is to connect all the dots using four straight lines. The solution to this puzzle is thought to be origin of the cliché *thinking outside the box*. This is due to the fact that the puzzle is solved by connecting dots with lines that go beyond the box limits. That was the premise adopted in the design of entanglement codes to connect blocks from the top and bottom borders of the helical lattice.

The second conversation was with my former advisor Prof. A. Nakao. During my master studies we discussed about important decisions in life. He mentioned that all paths are identical. There are many possible paths that connect the beginning to the end of life. All paths are like semi-circumferences connecting the poles of a sphere. In entanglement codes, two blocks die when all the paths (semi-circumferences) that connect them die.

The third conversation was with my husband R. Martin. My first approach was to connect blocks using a growing tree model but the results were not good. He asked me if I tried other connections on multiple directions. Suddenly, I visualized a grid.

Finally, the notion of redundancy propagation became promising and I used three parameters to control it. The parameter p controls the connections between top and bottom of the lattice, the parameter s control the length of the "semi-circumferences", and α controls the directions.

It is impossible to end this thesis without connecting our findings with Paul Baran's work. Paul Baran (1926-2011) was a pionner in the design of computer networks. He is well-known for his study on communications in distributed networks (Baran, 1964). He studied survivability for communications that occurred on a data transmission network composed of unreliable links. In his work, he argued that: “instead of using redundancy of coding we shall make use of redundancy of connectivity”. In this context, the redundancy level indicates the connectivity of a station (node) in a grid, in that sense, similar to the α parameter. He made the distinction between a perfect switching network in which all stations are connected and diversity of assignment, an alternative commonly used method. In diversity of assignment, a number of independent path are selected between each pair of stations that need to communicate reliably. He claimed the benefits of perfect switching network are passed on to the survivability of the system and conducted studies that supported his claims. He concluded that perfect switching provides an upper bound of expected system performance for a gridded network and that the bound is approached with redundancy levels of 3 or 4. We draw the attention of Baran’s work since what he argued about redundancy on distributed communications in 1964 has striking similarites to what we argue about redundancy on distributed storage in 2017.

Classic coding and replication behave like the diversity of assignment case. For many storage applications, these methods are useful, for instance, when systems need updates and do not want to keep a log. But long-term storage systems would benefit from entanglement codes. The outcomes of our study have witnessed attention from the academic environment and the industry. Entanglements can carry many implications to systems designs. We addressed many of them in the previous chapter. We foresee numerous opportunities and benefits in extending this study.

To conclude, this dissertation aims to give its reader multiple paths for studying and applying entanglement codes.

A Finding Erasure Patterns

This appendix extends the results given in Section 5.2.7. It provides more details about the method to find the smallest minimal erasure patterns and includes a table that characterizes fault tolerance for additional code settings. The table results are restricted to minimal erasure patterns that cause loss of up to 8 data nodes. $ME(1)$ is not considered, since all single-failures are repairable.

We assume that most, if not all, of the minimal erasure patterns formed with an even number of dead nodes fall into two categories:

1. Patterns that describe a connected subgraph inside of a region, namely *search space A*, constrained by *height* $= s$ and *width* $\leq p$. The area outside of the region is considered "safe", i.e., there are no missing elements connected to the subgraph. $ME(4)$ and $ME(6)$ shown in Figure 5.3 are straightforward examples for patterns that are found in search space A.
2. Patterns that describe a connected graph but do not cause data loss inside a safe region constrained by *height* $= s$ and *width* $\leq p-1$, namely *search space B*. The critical patterns presented in Figure 5.4 are examples for patterns found in this region.

This hypothesis helps us to reduce the feasible regions to conduct our search, both regions are illustrated in Figure A.1. The initial node d_i that fixes the relative position of the search space could be any valid top node in the lattice (except those located at the origin or end). Due to the regular lattice construction, the absolute position is irrelevant. The study of one section lattice is valid for the whole lattice.

Minimal patterns $ME(x)$ that are formed with an odd number x of dead nodes are more difficult to determine and the search regions may not help. In some cases, we succeeded by using $ME(x-1)$ as a baseline and adding a new node that is connected (at least) with α baseline nodes via distinct strands.

We chose the Prolog language for being a general purpose logic programming that allows us to

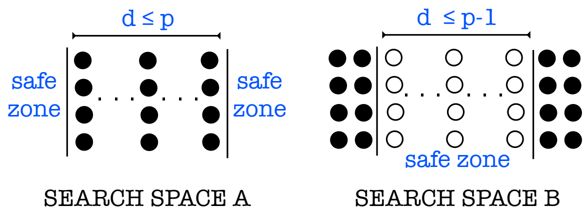


Figure A.1 – Feasible regions to search for MEs

Algorithm A.1 — Verify ME edge and vertex set

```

me_edge(S,P,L1,L2,[H]) :– H=X–Y
        , once(parity_loss(S,P,X,Y,_,L1,L2))
        , !
        .
me_edge(S,P,L1,L2,[H | T]) :– H=X–Y
        , once(parity_loss(S,P,X,Y,_,L1,L2))
        , me_edges(S,P,L1,L2,T)
        .
me_vertex(S,P,L1,L2,[H]) :– H=X
        , once(data_loss(S,P,X,L1,L2))
        , !
        .
me_vertex(S,P,L1,L2,[H | T]) :– H=X
        , once(data_loss(S,P,X,L1,L2))
        , me_vertex(S,P,L1,L2,T)
        .
```

implement very fast logic statements based on definitions presented in Chapter 5. Besides, the problem of finding patterns is well suited for backtracking algorithms, something that Prolog manages efficiently. A backtracking algorithm finds all or some solutions that satisfy certain constraints by incrementally adding candidates to a partial solution and abandonning candidates that cannot possibly lead to a valid solution.

Algorithm A.1 provides a mean to verify failure patterns found by visual inspections. In general terms, it checks if blocks are repairable recursively until all the elements in the pattern are recognized as repairable, or an irrecoverable failure pattern remains. We say that a data node d_k is locally recoverable if the repair process only uses the set of edges incident to the unavailable node without requiring a recursive call, and locally irrecoverable if not. A data object d_k is globally irrecoverable if the recursive repair is not successful. In other words, the repair fails for a set of locally irrecoverable data nodes, including d_k if they cannot propagate their redundant information to other available nodes that enable a successfull global repair. This set of irrecoverable nodes are a complex form (Definition 5.2.4).

In particular, Algorithm A.1 implements two backtracking rules to verify the list of edges *me_edge* and nodes *me_vertex* of a potential minimal erasure pattern. The backstracking rule *me_edge(S,P,L1,L2,[H|T])* requires as input parameters the code settings (s, p), the set of all

Input parameters	Description
S	Indicates the number of lattice rows, i.e., the code setting s.
P	Indicates the number of lattice columns, i.e., the code setting p.
X	Indicates the index i for data nodes d_i, or the left index i for edges $p_{i,j}$.
Y	Indicates the right index j for edges $p_{i,j}$.
atoms *{h, rh, lh}*	*h*, *rh* and *lh* indicate the strand. The rule operates from an initial position towards the end of the lattice. The initial position is considered the left index i of $p_{i,j}$. The rule outputs j when it was not given as parameter.
atoms *{hi, rhi, lhi}*	*hi*, *rhi* and *lhi* indicate the strand albeit the rule operates using inverse direction, from an initial position towards the beginning of the lattice. The initial position is considered the right index j of $p_{i,j}$. The rule outputs i when it was not given as parameter.
L1	Indicates the set of unavailable nodes.
L2	Indicates the set of unavailable edges.

Table A.1 – Input parameters for pattern searching rules.

nodes that are lost, $L1$, the set of all edges that are lost, $L2$, and the set of all edges potentially included in a ME,[H|T], for our purpose L2=[H|T]. The rule *me_vertex(S,P,L1,L2,[H|T])* requires the same input, except that the last list contains the set of vertices potentially belonging to a ME, for our purpose L1=[H|T]. The third and fourth inputs can contain a large set of unavailable elements that describes the state of the lattice. The rules output true if the elements listed in the input are necessary to define an erasure pattern. The backtracking rules maintain a list of potential solutions based on intermediate verifications using auxiliary rules. The rules are based on definitions presented in Chapter 5

The auxiliary rules given in Algorithm A.2 are rule *data_loss* based on Definition 5.2.1 (permanently erased node - dead node) and rules *parity_loss* based on Definition 5.2.2 (permanently erased edge). Rules *parity_loss* appear in lines 7, 11, and 15. Rule *data_loss* appears in lines 19 and is complemented with rules *tuple_loss* appearing in lines 24, 28, 32. Rules *tuple_loss* are more intermediate steps to verify if a pp-tuple that enables a locally repairable process is damaged, i.e., an edge is unavailable. More details on the input parameters are given in Table A.1. Note that the basic rule *edge* used by auxiliary rules is implemented in Algorithm 3.1 appearing in Chapter 4. The given code works for $\alpha = 3$ but it is easily adapted to $\alpha = 2$ by removing the rules that refer to the left-handed strand *lh* since only one helical strand suffices for $\alpha = 2$.

A.1 Example

We present an example that shows how to use Algorithm A.1 to verify a particular $ME(x)$. To examine $ME(6)$ for code settings ($\alpha = 3, s = 3, p = 3$), we explore the search space A and B.

Space A is constrained by *height* = 3 and *width* ≤ 3. Since the goal is to find the minimal erasure pattern inside the non-safe zone A, it make sense to reduce the space even more. We use space A defined with *height* = 3 and *width* = 2. In this reduced space there are exactly

Algorithm A.2 — Auxiliar rules for Algorithm A.1

```

loss(X,L1) :− memberchk(X,L1).
loss(X,Y,L2) :− memberchk(X−Y,L2).
double_loss(X,Y,right,L1,L2) :− loss(Y,L1); loss(X,Y,L2).
double_loss(X,Y,left,L1,L2) :− loss(Y,L1); loss(Y,X,L2).

parity_loss(S,_,X,Y,h,L1,L2) :− loss(X,Y,L2)
        , edge(S,_,X,W,hi), double_loss(W,X,right,L1,L2)
        , edge(S,_,Y,Z,h), double_loss(Z,Y,left,L1,L2)
        .
parity_loss(S,P,X,Y,rh,L1,L2) :− loss(X,Y,L2)
        , edge(S,P,X,W,rhi), double_loss(W,X,right,L1,L2)
        , edge(S,P,Y,Z,rh), double_loss(Z,Y,left,L1,L2)
        .
parity_loss(S,P,X,Y,lh,L1,L2) :− loss(X,Y,L2)
        , edge(S,P,X,W,lhi), double_loss(W,X,right,L1,L2)
        , edge(S,P,Y,Z,lh), double_loss(Z,Y,left,L1,L2)
        .
data_loss(S,P,X,L1,L2) :− loss(X,L1),
        , tuple_loss(S,_,X,h,L1,L2)
        , tuple_loss(S,P,X,rh,L1,L2)
        , tuple_loss(S,P,X,lh,L1,L2)
        .
tuple_loss(S,_,X,h,L1,L2) :−
        edge(S,_,X,Y,h), parity_loss(S,_,X,Y,h,L1,L2)
        ; edge(S,_,X,Z,hi), parity_loss(S,_,Z,X,h,L1,L2)
        .
tuple_loss(S,P,X,rh,L1,L2) :−
        edge(S,P,X,Y,rh), parity_loss(S,P,X,Y,rh,L1,L2)
        ; edge(S,P,X,Z,rhi), parity_loss(S,P,Z,X,rh,L1,L2)
        .
tuple_loss(S,P,X,lh,L1,L2) :−
        edge(S,P,X,Y,lh), parity_loss(S,P,X,Y,lh,L1,L2)
        ; edge(S,P,X,Z,lhi), parity_loss(S,P,Z,X,lh,L1,L2)
        .
```

six nodes. The inital position in the lattice, say node d_{22}, is arbitrarily chosen. The induced subgraph $S = \{V, E\}$ constructed with all the nodes inside the non-safe zone is our minimal erasure pattern candidate:

$$V = \{d_{22}, d_{23}, d_{24}, d_{25}, d_{26}, d_{27}\}$$
$$E = \{p_{22,25}, p_{23,26}, p_{24,27}, p_{22,26}, p_{23,27}, p_{24,25}, p_{22,27}, p_{23,25}, p_{24,26}\}$$

To verify this pattern using Algorithm A.1 we adapt nodes and edges indexes notation. Vertices are indicated with the index only, e.g., 22. Edges are indicated with X-Y, e.g., 22-25. Input lists are enclosed in square brackets. The order of elements in the list is irrelevant. Now, we can call the rules (where V and E are replaced by the lists described above):

```
me_vertex(3,3,V,E,V).
me_edge(3,3,V,E,E).
```

Each rule outputs true if the last parameter contains the list of elements that are necessary to define a minimal pattern. The rules verify necessary conditions, not sufficient. If we remove one of the elements, the output remains true. On the contrary, if we add an element that is not required to define an erasure pattern the output will be false.

Using space A, we found for the setting ($\alpha = 3, s = 3, p = 3$) that $ME(6) \overset{A}{=} 15$ (the total number of elements in both sets).

To test inside space B, we use a subgraph similar to the examples showed in Figure 5.4. As it was shown in the figure, this pattern is formed by a sequence of $x/2$ dead nodes, followed by $p - x/2$ live nodes and the other set of $x/2$ dead nodes, all aligned consecutively in the same strand and their corresponding edge connections that connect the dead nodes building an erasure pattern. In this case, the safe zone is defined by the live nodes located in the middle of the sequence. An example is given here:

$$V = \{d_{22}, d_{25}, d_{28}, d_{31}, d_{34}, d_{37}\}$$
$$E = \{p_{22,25}, p_{25,28}, p_{31,34}, p_{34,37}, p_{22,26}, p_{26,30}, p_{30,31}, p_{25,29}, p_{29,33}, p_{33,34}, p_{28,32}, p_{32,36}, p_{36,37}, p_{22,27}, p_{27,29}, p_{29,31}, p_{25,30}, p_{30,32}, p_{32,34}, p_{28,33}, p_{33,35}, p_{35,37}\}$$

Since the live nodes are not considered in the definition of a minimal erasure pattern, they are not listed in the vertex set. The rules validate that the sets are a minimal erasure pattern for $ME(6)$. But, in this case, $ME(6) \overset{B}{=} 28$. Clearly, the smallest failure pattern is the one found in the first search. Noteworthy, the size corresponds to the lower bound for redundancy propagation given in Theorem 5.2.2. The existence of a smaller subgraph that cause the loss of six data nodes inside the space A, built with code AE(3,3,3), is impossible.

A.2 Results

This section presents Table A.2 that summarizes our results. The table provides the size for the minimal failure patterns found using the finding patterns method described in Section A.1. It illustrates the irregularity of fault tolerance in a helical lattice. It also unveils some regularities listed here:

Lower bounds. If $s = p$, the pattern $ME(x)$ with $x = 2 \cdot s$ determines the lower possible fault-tolerance. Although the code can provide more fault-tolerance when p increases, this setting cannot be judged as good or bad per se. Trade-offs between performance and reliability should be considered to choose the best parameters. More details can be found in Section 7.3.6.

Fault-tolerance. Generally, fault-tolerance increases as s and p increase except for the upper bound cases.

Upper bounds. There are some patterns that appear due to the high connectivity between nodes. For example, the $ME(8)$ shown in Figure 5.3 is an upper bound. Note that its size cannot be modified by increasing s or p. The only way to overcome the upper bound and increase fault-tolerance is to increase α.

Safe zones. Some $ME(x)$ found in space B can be used as a baseline to find $ME(y)$ with $y > x$. For those cases, we do not restrict the search space anymore. We try to accommodate the extra $y - x$ nodes in a valid pattern that shares the largest number of edges.

Live nodes. Patterns that include live nodes result in larger $ME(x)$s. Hence, patterns found in space B are larger than the lower bound due to the appearance of live nodes, i.e., making the distance between dead nodes larger.

	$ME(x)$ size, with x data loss						
	$ME(2)$	$ME(3)$	$ME(4)$	$ME(5)$	$ME(6)$	$ME(7)$	$ME(8)$
				$\alpha = 1$			
REF.	**3**	**5**	**7**	**9**	**11**	**13**	**15**
$(1, -)$	3	5	7	9	11	13	15
				$\alpha = 2$			
REF.	**4**	**6**	**8**	**10**	**12**	**14**	**16**
$(2,2)$	6	11	**8**	11	13	16	18
$(2,3)$	7	13	**8**	12	13	16	18
$(2,4)$	8	15	**8**	13	13	17	18
$(2,5)$	9	17	**8**	14	13	17	18
$(3,3)$	8	15	**8**	13	**12**	15	18
$(3,4)$	9	17	**8**	14	13	18	18
$(3,5)$	10	19	**8**	15	13	19	18
$(4,4)$	10	19	**8**	15	13	19	**16**
$(5,5)$	12	23	**8**	17	13	21	18
				$\alpha = 3$			
REF.	**5**	**9**	**10**	**14**	**15**	**19**	**20**
$(2,2)$	8	10	**10**	**14**	17	20	22
$(2,3)$	9	17	14	21	22	21	22
$(2,4)$	10	19	14	22	22	29	22
$(2,5)$	11	21	14	23	22	30	22
$(3,3)$	11	21	18	27	**15**	**19**	**20**
$(3,4)$	12	13	18	28	21	23	**20**
$(3,5)$	13	25	18	29	21	23	**20**
$(4,4)$	14	27	22	34	22	23	**20**
$(5,5)$	17	33	26	41	29	23	**20**

Table A.2 – Fault tolerance for various code settings (s, p). The table shows the size of minimal erasure patterns |ME(x)| with $x \in [2,8]$ for valid settings of single-, double-, and triple-entanglements with $s \in [2,3]$ and $p \in [2,5]$. The reference values (REF.) indicate the lower bound for a particular α.

Bibliography

Adams, I., Miller, E. L., and Rosenthal, D. S. (2011). Using storage class memory for archives with dawn, a durable array of wimpy nodes. Technical report, Technical Report UCSC-SSRC-11-07, University of California, Santa Cruz.

Adya, A., Bolosky, W. J., Castro, M., Cermak, G., Chaiken, R., Douceur, J. R., Howell, J., Lorch, J. R., Theimer, M., and Wattenhofer, R. P. (2002). Farsite: Federated, available, and reliable storage for an incompletely trusted environment. *ACM SIGOPS Operating Systems Review*, 36(SI):1–14.

Amer, A., Long, D. D., Schwarz, T., and Paris, J.-F. (2008). Increased reliability with sspiral data layouts. In *2008 IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*, pages 1–10. IEEE.

Anderson, R. et al. (1996). The eternity service. In *Proceedings of PRAGOCRYPT*, volume 96, pages 242–252.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2010). A view of cloud computing. *Communications of the ACM*, 53(4):50–58.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Aspnes, J., Feigenbaum, J., Yampolskiy, A., and Zhong, S. (2007). Towards a theory of data entanglement. *Theoretical Computer Science*, 389(1):26–43.

Ateniese, G., Dagdelen, O., Damgård, I., and Venturi, D. (2016). Entangled cloud storage. *Future Gener. Comput. Syst.*, 62(C):104–118.

Bailis, P. and Kingsbury, K. (2014). The network is reliable: An informal survey of real-world communications failures. ACM Queue 12(7), July 2014. Also appears in Communications of the ACM 57(9):48-55, September 2014.

Bairavasundaram, L. N., Goodson, G. R., Pasupathy, S., and Schindler, J. (2007). An analysis of latent sector errors in disk drives. In *ACM SIGMETRICS Performance Evaluation Review*, volume 35, pages 289–300. ACM.

Baran, P. (1964). On distributed communications networks. *IEEE transactions on Communications Systems*, 12(1):1–9.

Bardini, T. (2017). A. m. Turing awards: Douglas engelbart. ACM Website.

Basak, J., Wadhwani, K., and Voruganti, K. (2016). Storage workload identification. *Trans. Storage*, 12(3):14:1–14:30.

Beach, B. (2014). What hard drive should i buy? Backblaze Blog from January.

Beach, B. (2015). Backblaze open sources reed-solomon erasure coding source code.

Benedetto, S. and Montorsi, G. (1996). Design of parallel concatenated convolutional codes. *IEEE Transactions on Communications*, 44(5):591–600.

Berrou, C. and Glavieux, A. (1996). Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Transactions on communications*, 44(10):1261–1271.

Berrou, C. and Glavieux, A. (2003). Turbo codes. *Encyclopedia of Telecommunications.*

Berrou, C., Glavieux, A., and Thitimajshima, P. (1993). Near shannon limit error-correcting coding and decoding: Turbo-codes. 1. In *Communications, 1993. ICC'93 Geneva. Technical Program, Conference Record, IEEE International Conference on*, volume 2, pages 1064–1070. IEEE.

Bessani, A., Correia, M., Quaresma, B., André, F., and Sousa, P. (2013). Depsky: dependable and secure storage in a cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):12.

Blake, C. and Rodrigues, R. (2003). High availability, scalable storage, dynamic peer networks: Pick two. In *HotOS*, volume 3, page 1.

Blaum, M., Brady, J., Bruck, J., and Menon, J. (1995). Evenodd: An efficient scheme for tolerating double disk failures in raid architectures. *IEEE Transactions on computers*, 44(2):192–202.

Bornholt, J., Lopez, R., Carmean, D. M., Ceze, L., Seelig, G., and Strauss, K. (2016). A dna-based archival storage system. *ACM SIGOPS Operating Systems Review*, 50(2):637–649.

Braam, P. J. and Zahir, R. (2002). Lustre: A scalable, high performance file system. *Cluster File Systems, Inc.*

Brewer, E., Ying, L., Greenfield, L., and Cypher, R. (2016). Disks for data centers. Technical report, Google Research.

Brewster, K. (2016). Help us keep the archive free, accessible, and reader private. https://blog.archive.org/2016/11/29/help-us-keep-the-archive-free-accessible-and-private/.

Burkhard, W. A. and Menon, J. (1993). Disk array storage system reliability. In *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*, pages 432–441. IEEE.

Cachin, C., Keidar, I., and Shraer, A. (2009). Trusting the cloud. *Acm Sigact News*, 40(2):81–86.

Cappello, F., Geist, A., Gropp, W., Kale, S., Kramer, B., and Snir, M. (2014). Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1):5–28.

Carretero, J. (2017). Hpc and big data convergence. Lecture at CUSO Winter School in Computer Science, Champery, Switzerland.

Chansler, R. J. (2012). Data availability and durability with the hadoop distributed file system. *The USENIX Magzine*.

Chatzigeorgiou, I., Rodrigues, M. R., Wassell, I. J., and Carrasco, R. A. (2009). Analysis and design of punctured rate-1/2 turbo codes exhibiting low error floors. *IEEE Journal on Selected Areas in Communications*, 27(6).

Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., and Patterson, D. A. (1994). Raid: High-performance, reliable secondary storage. *ACM Computing Surveys (CSUR)*, 26(2):145–185.

Choi, H. and Varian, H. (2012). Predicting the present with google trends. *Economic Record*, 88(s1):2–9.

Christidis, K. and Devetsikiotis, M. (2016). Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303.

Chun, B.-G., Dabek, F., Haeberlen, A., Sit, E., Weatherspoon, H., Kaashoek, M. F., Kubiatowicz, J., and Morris, R. (2006). Efficient replica maintenance for distributed storage systems. In *NSDI*, volume 6, pages 4–4.

Church, G. M., Gao, Y., and Kosuri, S. (2012). Next-generation digital information storage in dna. *Science*, 337(6102):1628–1628.

Cidon, A., Rumble, S. M., Stutsman, R., Katti, S., Ousterhout, J. K., and Rosenblum, M. (2013). Copysets: Reducing the frequency of data loss in cloud storage. In *USENIX Annual Technical Conference*, pages 37–48.

Clarke, I., Sandberg, O., Wiley, B., and Hong, T. W. (2001). Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer.

Colarelli, D. and Grunwald, D. (2002). Massive arrays of idle disks for storage archives. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–11. IEEE Computer Society Press.

Colbourn, C. J. (1993). Analysis and synthesis problems for network resilience. *Mathematical and computer modelling*, 17(11):43–48.

Corbett, P., English, B., Goel, A., Grcanac, T., Kleiman, S., Leong, J., and Sankar, S. (2004). Row-diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, pages 1–14.

Corderí, I., Schwarz, T., Amer, A., Long, D. D., and Pâris, J.-F. (2010). Self-adjusting two-failure tolerant disk arrays. In *Petascale Data Storage Workshop (PDSW), 2010 5th*, pages 1–5. IEEE.

Dabek, F., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. (2001). Wide-area cooperative storage with cfs. *SIGOPS Oper. Syst. Rev.*, 35(5):202–215.

Dean, J. (2010). Evolution and future directions of large-scale storage and computation systems at google. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 1–1, New York, NY, USA. ACM.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., and Vogels, W. (2007). Dynamo: amazon's highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220.

Dimakis, A. G., Godfrey, P. B., Wu, Y., Wainwright, M. J., and Ramchandran, K. (2010). Network coding for distributed storage systems. *IEEE Transactions on Information Theory*, 56(9):4539–4551.

Dimakis, A. G., Prabhakaran, V., and Ramchandran, K. (2006). Decentralized erasure codes for distributed networked storage. *IEEE/ACM Transactions on Networking (TON)*, 14(SI):2809–2816.

Dimakis, A. G., Ramchandran, K., Wu, Y., and Suh, C. (2011). A survey on network codes for distributed storage. *Proceedings of the IEEE*, 99(3):476–489.

Dingledine, R., Freedman, M. J., and Molnar, D. (2001). The free haven project: Distributed anonymous storage service. In *Designing Privacy Enhancing Technologies*, pages 67–95. Springer.

Dongarra, J., Tourancheau, B., and Cappello, F. (2009a). Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities. *The International Journal of High Performance Computing Applications*, 23(3):212–226.

Dongarra, J., Tourancheau, B., and Plank, J. S. (2009b). The raid-6 liber8tion code. *The International Journal of High Performance Computing Applications*, 23(3):242–251.

Drago, I., Bocchi, E., Mellia, M., Slatman, H., and Pras, A. (2013). Benchmarking personal cloud storage. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 205–212. ACM.

Espinal, X., Adde, G., Chan, B., Iven, J., Presti, G. L., Lamanna, M., Mascetti, L., Pace, A., Peters, A., Ponce, S., et al. (2014). Disk storage at cern: Handling lhc data and beyond. In *Journal of Physics: Conference Series*, volume 513, page 042017. IOP Publishing.

Estrada-Galinanes, V., , and Pâris, J.-F. (2017a). Spigot codes. *(submitted for publication)*, page to appear.

Estrada Galinanes, V. (2013). Towards eternal storage using data entanglement. Technical report, Institut f¸r Informatik und angewandte Mathematik.

Estrada Galinanes, V. (2015). The entangled storage project: Building a trustworthy long-term data storage. Technical report, Institut f¸r Informatik und angewandte Mathematik.

Estrada-Galinanes, V. (2016). Trustworthy entangled storage. Eurosys Doctoral Workshop.

Estrada-Galinanes, V. (2017). A research playground for examining erasure codes. In *Proceedings of the 10th ACM International Systems and Storage Conference*, New York, NY, USA. ACM.

Estrada-Galinanes, V. and Felber, P. (2013). Helical entanglement codes: An efficient approach for designing robust distributed storage systems. In *Symposium on Self-Stabilizing Systems*, pages 32–44. Springer.

Estrada-Galinanes, V. and Felber, P. (2015a). Changing the redundancy paradigm: Challenges of building an entangled storage. Usenix FAST - Work in Progress Session.

Estrada-Galinanes, V. and Felber, P. (2015b). Ensuring data durability with increasingly interdependent content. In *2015 IEEE International Conference on Cluster Computing*, pages 162–165. IEEE.

Estrada-Galinanes, V. and Felber, P. (2016). Override raid: Redundant arrays of interdependent disks. Usenix FAST - Work in Progress Session.

Estrada-Galinanes, V., Miller, E., and Felber, P. (2017b). Alpha entanglement codes: Practical erasure codes to archive big data in unreliable environments. *(submitted for publication)*, page to appear.

Estrada-Galinanes, V., Pâris, J.-F., and Felber, P. (2016). Simple data entanglement layouts with high reliability. In *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*.

Fan, B., Tantisiriroj, W., Xiao, L., and Gibson, G. (2009). Diskreduce: Raid for data-intensive scalable computing. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, pages 6–10. ACM.

Fan, B., Tantisiriroj, W., Xiao, L., and Gibson, G. (2011). Diskreduce: Replication as a prelude to erasure coding in data-intensive scalable computing. *SC11*.

Faraci, V. (2006). Calculating failure rates of series/parallel networks. *The Journal of Alion" s, System Reliability Center.*

Feldman, J., Abou-Faycal, I., and Frigo, M. (2002). A fast maximum-likelihood decoder for convolutional codes. In *Vehicular Technology Conference, 2002. Proceedings. VTC 2002-Fall. 2002 IEEE 56th*, volume 1, pages 371–375. IEEE.

Ford, D., Labelle, F., Popovici, F. I., Stokely, M., Truong, V.-A., Barroso, L., Grimes, C., and Quinlan, S. (2010). Availability in globally distributed storage systems. In *OSDI*, pages 61–74.

Fragouli, C., Le Boudec, J.-Y., and Widmer, J. (2006). Network coding: an instant primer. *ACM SIGCOMM Computer Communication Review*, 36(1):63–68.

Fragouli, C., Soljanin, E., et al. (2008). Network coding applications. *Foundations and Trends® in Networking*, 2(2):135–269.

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

Gang, W., Xiaoguang, L., Sheng, L., Guangjun, X., and Jing, L. (2008). Generalizing rdp codes using the combinatorial method. In *Network Computing and Applications, 2008. NCA'08. Seventh IEEE International Symposium on*, pages 93–100. IEEE.

Ghemawat, S., Gobioff, H., and Leung, S.-T. (2003). The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43.

Gkantsidis, C. and Rodriguez, P. R. (2005). Network coding for large scale content distribution. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 4, pages 2235–2245. IEEE.

Goddard, W. and Lynott, J. (1970). Direct access magnetic disc storage device. US Patent 3,503,060.

Goldman, N., Bertone, P., Chen, S., Dessimoz, C., LeProust, E. M., Sipos, B., and Birney, E. (2013). Towards practical, high-capacity, low-maintenance information storage in synthesized dna. *Nature*, 494(7435):77–80.

Greenan, K., Miller, E. L., and Wylie, J. (2008a). Reliability of flat xor-based erasure codes on heterogeneous devices. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2008)*, pages 147–156.

Greenan, K. M., Li, X., and Wylie, J. J. (2010). Flat xor-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–14. IEEE.

Greenan, K. M., Miller, E. L., and Schwarz, S. T. J. (2008b). Optimizing galois field arithmetic for diverse processor architectures and applications. In *Modeling, Analysis and Simulation of Computers and Telecommunication Systems, 2008. MASCOTS 2008. IEEE International Symposium on*, pages 1–10. IEEE.

Greenan, K. M., Miller, E. L., and Schwarz, T. J. (2007a). Analysis and construction of galois fields for efficient storage reliability. *Tech. Rep.*

Greenan, K. M., Miller, E. L., Schwarz, T. J., and Long, D. D. (2007b). Disaster recovery codes: increasing reliability with large-stripe erasure correcting codes. In *Proceedings of the 2007 ACM workshop on Storage security and survivability*, pages 31–36. ACM.

Ha, J. and McLaughlin, S. W. (2003). Optimal puncturing of irregular low-density parity-check codes. In *Communications, 2003. ICC'03. IEEE International Conference on*, volume 5, pages 3110–3114. IEEE.

Haeberlen, A., Mislove, A., and Druschel, P. (2005). Glacier: Highly durable, decentralized storage despite massive correlated failures. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 143–158. USENIX Association.

Hafner, J. L. (2005). Weaver codes: Highly fault tolerant erasure codes for storage systems. In *FAST*, volume 5, pages 16–16.

Hafner, J. L. (2006). Hover erasure codes for disk arrays. In *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pages 217–226. IEEE.

Hafner, J. L. and Rao, K. (2006). Notes on reliability models for non-mds erasure codes. *IBM Res. rep. RJ10391*.

Hagmann, R. (1987). Reimplementing the cedar file system using logging and group commit. *SIGOPS Oper. Syst. Rev.*, 21(5):155–162.

Handbook, E. R. D. (1982). Mil-hdbk-338b, october 1998. *Robert G. Arno received his BS in Electrical Engineering from State University of New York at Utica/Rome in.*

Hellerstein, L., Gibson, G. A., Karp, R. M., Katz, R. H., and Patterson, D. A. (1994). Coding techniques for handling failures in large disk arrays. *Algorithmica*, 12(2-3):182–208.

Holland, M. and Gibson, G. A. (1992). Parity declustering for continuous operation in redundant disk arrays. *SIGPLAN Not.*, 27(9):23–35.

Huan, W. and Nakazato, H. (2015). Failure detection in p2p-grid system. *IEICE TRANSACTIONS on Information and Systems*, 98(12):2123–2131.

Huang, C., Chen, M., and Li, J. (2007). Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*, pages 79–86. IEEE.

Huang, C., Chen, M., and Li, J. (2013). Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *ACM Transactions on Storage (TOS)*, 9(1):3.

Huang, C., Simitci, H., Xu, Y., Ogus, A., Calder, B., Gopalan, P., Li, J., and Yekhanin, S. (2012). Erasure coding in windows azure storage. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 15–26.

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

Jaffe, E. and Kirkpatrick, S. (2009). Architecture of the internet archive. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, page 11. ACM.

Jeon, S. (2010). *Low-Density Parity-Check Codes for Data Storage and Memory Systems.* PhD thesis, Carnegie Mellon University Pittsburgh, PA.

Jiekak, S., Kermarrec, A.-M., Le Scouarnec, N., Straub, G., and Van Kempen, A. (2013). Regenerating codes: A system perspective. *ACM SIGOPS Operating Systems Review*, 47(2):23–32.

Kamra, A., Misra, V., Feldman, J., and Rubenstein, D. (2006). Growth codes: Maximizing sensor network data persistence. *SIGCOMM Comput. Commun. Rev.*, 36(4):255–266.

Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., and Lewin, D. (1997). Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM.

Khan, O., Burns, R. C., Plank, J. S., and Huang, C. (2011). In search of i/o-optimal recovery from disk failures. In *HotStorage*.

Khan, O., Burns, R. C., Plank, J. S., Pierce, W., and Huang, C. (2012). Rethinking erasure codes for cloud file systems: minimizing i/o for recovery and degraded reads. In *FAST*, page 20.

Kosba, A., Miller, A., Shi, E., Wen, Z., and Papamanthou, C. (2016). Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 839–858. IEEE.

Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., et al. (2000). Oceanstore: An architecture for global-scale persistent storage. *ACM Sigplan Notices*, 35(11):190–201.

Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40.

Lee, W. S. (2004). Two-dimensional storage array with prompt parity in one dimension and delayed parity in a second dimension. US Patent 6,675,318.

Li, D. and Haimes, Y. Y. (1992). A decomposition method for optimization of large-system reliability. *IEEE Transactions on Reliability*, 41(2):183–188.

Li, M., Qin, C., Lee, P. P., and Li, J. (2014). Convergent dispersal: Toward storage-efficient security in a cloud-of-clouds. In *HotCloud*.

Li, P., Li, J., Stones, R. J., Wang, G., Li, Z., and Liu, X. (2016). Procode: A proactive erasure coding scheme for cloud storage systems. In *Reliable Distributed Systems (SRDS), 2016 IEEE 35th Symposium on*, pages 219–228. IEEE.

Lillibridge, M., Elnikety, S., Birrell, A., Burrows, M., and Isard, M. (2003). A cooperative internet backup scheme. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 3–3. USENIX Association.

Lin, W., Chiu, D. M., and Lee, Y. (2004). Erasure code replication revisited. In *Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on*, pages 90–97. IEEE.

Luby, M. (2002). Lt codes. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*, FOCS '02, pages 271–, Washington, DC, USA. IEEE Computer Society.

Luby, M., Shokrollahi, A., Watson, M., Stockhammer, T., and Minder, L. (2011). Proposed standard rfc 6330 - raptorq forward error correction scheme for object delivery. Technical report, Internet Engineering Task Force.

Lucky, R. W. (1993). *Lucky Strikes... Again:(Feats and Foibles of Engineers)*. John Wiley & Sons.

Ma, A., Traylor, R., Douglis, F., Chamness, M., Lu, G., Sawyer, D., Chandra, S., and Hsu, W. (2015). Raidshield: Characterizing, monitoring, and proactively protecting against disk failures. *Trans. Storage*, 11(4):17:1–17:28.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

MacKay, D. J. (2005). Fountain codes. *IEE Proceedings-Communications*, 152(6):1062–1068.

MacKay, D. J., Wilson, S. T., and Davey, M. C. (1999). Comparison of constructions of irregular gallager codes. *IEEE Transactions on Communications*, 47(10):1449–1454.

Mager, T., Biersack, E., and Michiardi, P. (2012). A measurement study of the wuala on-line storage service. In *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*, pages 237–248. IEEE.

Maniatis, P., Roussopoulos, M., Giuli, T. J., Rosenthal, D. S., and Baker, M. (2005). The lockss peer-to-peer digital preservation system. *ACM Transactions on Computer Systems (TOCS)*, 23(1):2–50.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Mercier, H., Augier, M., and Lenstra, A. K. (2015). Step-archival: Storage integrity and anti-tampering using data entanglement. In *Information Theory (ISIT), 2015 IEEE International Symposium on*, pages 1590–1594. IEEE.

Miranda, A., Effert, S., Kang, Y., Miller, E. L., Popov, I., Brinkmann, A., Friedetzky, T., and Cortes, T. (2014). Random slicing: Efficient and scalable data placement for large-scale storage systems. *ACM Transactions on Storage*, 10(3).

Muralidhar, S., Lloyd, W., Roy, S., Hill, C., Lin, E., Liu, W., Pan, S., Shankar, S., Sivakumar, V., Tang, L., et al. (2014). f4: Facebook's warm blob storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 383–398.

Nagaraj, S. V. (2017). Review of algebraic coding theory revised edition by elwyn berlekamp. *SIGACT News*, 48(1):23–26. Reviewer-Nagaraj, S. V.

Oggier, F. (2013). On coding techniques for networked distributed storage systems. Lecture at the First European Training School on Network Coding, ICT Cost Action IC1104.

Oggier, F. and Datta, A. (2011). Self-repairing homomorphic codes for distributed storage systems. In *INFOCOM, 2011 Proceedings IEEE*, pages 1215–1223. IEEE.

Panzer-Steindel, B. (2007). Data integrity. *CERN/IT*.

Papailiopoulos, D. S. and Dimakis, A. G. (2014). Locally repairable codes. *IEEE Transactions on Information Theory*, 60(10):5843–5855.

Patterson, D. A., Gibson, G., and Katz, R. H. (1988). A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, SIGMOD '88, pages 109–116, New York, NY, USA. ACM.

Peters, A. J. and Janyst, L. (2011). Exabyte scale storage at cern. In *Journal of Physics: Conference Series, Part 5: Computing Fabrics and Networking Technologies*, volume 331. IOP Publishing.

Pike, R., Presotto, D., Dorward, S., Flandrena, B., Thompson, K., Trickey, H., and Winterbottom, P. (1995). Plan 9 from bell labs. *Computing systems*, 8(2):221–254.

Pinheiro, E., Weber, W.-D., and Barroso, L. A. (2007). Failure trends in a large disk drive population. In *FAST*, volume 7, pages 17–23.

Pishro-Nik, H. and Fekri, F. (2007). Results on punctured low-density parity-check codes and improved iterative decoding techniques. *IEEE Transactions on Information Theory*, 53(2):599–614.

Plank, J. S. et al. (1997). A tutorial on reed-solomon coding for fault-tolerance in raid-like systems. *Softw., Pract. Exper.*, 27(9):995–1012.

Plank, J. S., Greenan, K. M., and Miller, E. L. (2013). Screaming fast galois field arithmetic using intel simd instructions. In *FAST*, pages 299–306.

Plank, J. S. and Thomason, M. G. (2003). On the practical use of ldpc erasure codes for distributed storage applications. *Technical Report CS-03–510*.

Plank, J. S. and Xu, L. (2006). Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. In *Network Computing and Applications, 2006. NCA 2006. Fifth IEEE International Symposium on*, pages 173–180. IEEE.

Protocol Labs (2014). Interplanetary file system (ipfs - the permanent web). https://github.com/ipfs/ipfs.

Quinlan, S. and Dorward, S. (2002). Venti: A new approach to archival storage. In *FAST*, volume 2, pages 89–101.

Raab, M. and Steger, A. (1998). “balls into bins”—a simple and tight analysis. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 159–170. Springer.

Rashmi, K. V., Shah, N. B., and Kumar, P. V. (2011). Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction. *IEEE Transactions on Information Theory*, 57(8):5227–5239.

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

Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J., et al. (2004). Handling churn in a dht. In *Proceedings of the USENIX Annual Technical Conference*, volume 6, pages 127–140. Boston, MA, USA.

Richardson, T. J., Shokrollahi, M. A., and Urbanke, R. L. (2001). Design of capacity-approaching irregular low-density parity-check codes. *IEEE transactions on information theory*, 47(2):619–637.

Richardson, T. J. and Urbanke, R. L. (2001). Efficient encoding of low-density parity-check codes. *IEEE Transactions on Information Theory*, 47(2):638–656.

Ripeanu, M. (2001). Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100. IEEE.

Riska, A. and Riedel, E. (2006). Disk drive level workload characterization. In *Proceedings of the annual conference on USENIX’06 Annual Technical Conference*, pages 9–9. USENIX Association.

Rosenblum, M. and Ousterhout, J. K. (1992). The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems (TOCS)*, 10(1):26–52.

Rosenthal, A. (1977). Computing the reliability of complex networks. *SIAM Journal on Applied Mathematics*, 32(2):384–393.

Rosenthal, D. S., Robertson, T. S., Lipkis, T., Reich, V., and Morabito, S. (2005). Requirements for digital preservation systems: A bottom-up approach. *arXiv preprint cs/0509018*.

Rosenthal, D. S., Rosenthal, D. C., Miller, E. L., Adams, I. F., Storer, M. W., and Zadok, E. (2012). The economics of long-term digital storage. *Memory of the World in the Digital Age, Vancouver, BC*.

Rozenberg, G. (1997). *Handbook of graph grammars and computing by graph transformation*, volume 1. World Scientific.

Rozier, E. W., Belluomini, W., Deenadhayalan, V., Hafner, J., Rao, K., and Zhou, P. (2009). Evaluating the impact of undetected disk errors in raid systems. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 83–92. IEEE.

Santry, D. S., Feeley, M. J., Hutchinson, N. C., Veitch, A. C., Carton, R. W., and Ofir, J. (2000). Deciding when to forget in the elephant file system. *SIGOPS Oper. Syst. Rev.*, 34(2):18–19.

Sathiamoorthy, M., Asteris, M., Papailiopoulos, D., Dimakis, A. G., Vadali, R., Chen, S., and Borthakur, D. (2013). Xoring elephants: Novel erasure codes for big data. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 325–336. VLDB Endowment.

Schroeder, B. and Gibson, G. (2010). A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing*, 7(4):337–350.

Schroeder, B. and Gibson, G. A. (2007). Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *FAST*, volume 7, pages 1–16.

Schwarz, T., Amer, A., Kroeger, T., Miller, E. L., Long, D. D. E., and Pâris, J.-F. (2016). RESAR: Reliable storage at exabyte scale. In *Proceedings of the 24th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2016)*.

Schwarz, T. J. and Burkhard, W. A. (1992). Raid organization and performance. In *ICDCS*, pages 318–325.

Schwarz, T. J., Xin, Q., Miller, E. L., Long, D. D., Hospodor, A., and Ng, S. (2004). Disk scrubbing in large archival storage systems. In *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings. The IEEE Computer Society's 12th Annual International Symposium on*, pages 409–418. IEEE.

Schwarz, T. J. E. (1994). *Reliability and performance of disk arrays.* PhD thesis, University of California, San Diego.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423.

Shannon, C. E. (1958). Von neumann's contributions to automata theory. *Bull. Amer. Math. Soc*, 64:123–129.

Shiroishi, Y., Fukuda, K., Tagawa, I., Iwasaki, H., Takenoiri, S., Tanaka, H., Mutoh, H., and Yoshikawa, N. (2009). Future options for hdd storage. *IEEE Transactions on Magnetics*, 45(10):3816–3822.

Shokrollahi, A. (2006). Raptor codes. *IEEE transactions on information theory*, 52(6):2551–2567.

Shokrollahi, A., Luby, M., Watson, M., and Stockhammer, T. (2007). Proposed standard rfc 5053 - raptor forward error correction scheme for object delivery. Technical report, Internet Engineering Task Force.

Shooman, M. L. (1990). *Probabilistic Reliability: An Engineering Approach*. Krieger, Melbourne, FL.

Shooman, M. L. (2003). *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design*. John Wiley & Sons.

Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010). The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. IEEE.

Slamanig, D. and Hanser, C. (2012). On cloud storage and the cloud of clouds approach. In *Internet Technology And Secured Transactions, 2012 International Conference for*, pages 649–655. IEEE.

Smith, B. P., Farhood, A., Hunt, A., Kschischang, F. R., and Lodge, J. (2012). Staircase codes: Fec for 100 gb/s otn. *Journal of Lightwave Technology*, 30(1):110–117.

Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160.

Storer, M. W., Greenan, K. M., Miller, E. L., and Voruganti, K. (2009). Potshards—a secure, recoverable, long-term archival storage system. *ACM Transactions on Storage (TOS)*, 5(2):5.

Stubblefield, A. and Wallach, D. S. (2001). Dagster: censorship-resistant publishing without replication. *Rice University, Dept. of Computer Science, Tech. Rep. TR01-380*.

Suh, C. and Ramchandran, K. (2009). Exact regeneration codes for distributed storage repair using interference alignment. *arXiv preprint arXiv:1001.0107*.

Tang, H., Gulbeden, A., Zhou, J., Strathearn, W., Yang, T., and Chu, L. (2004). A self-organizing storage cluster for parallel data-intensive applications. In *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference*, pages 52–52. IEEE.

Tanner, R. (1981). A recursive approach to low complexity codes. *IEEE Transactions on information theory*, 27(5):533–547.

Tran, D. N., Chiang, F., and Li, J. (2008). Friendstore: cooperative online backup using trusted nodes. In *Proceedings of the 1st Workshop on Social Network Systems*, pages 37–42. ACM.

Turing, A. M. (1937). On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265.

Vardy, A. (1997). Algorithmic complexity in coding theory and the minimum distance problem. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 92–109. ACM.

Vermij, E., Fiorin, L., Hagleitner, C., and Bertels, K. (2014). Exascale radio astronomy: Can we ride the technology wave? In *International Supercomputing Conference*, pages 35–52. Springer.

Vishwanath, K. V. and Nagappan, N. (2010). Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 193–204. ACM.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269.

Von Neumann, J. (1956). Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98.

Von Neumann, J. (1993). First draft of a report on the edvac. *IEEE Annals of the History of Computing*, 15(4):27–75.

Vrable, M., Savage, S., and Voelker, G. M. (2012). Bluesky: A cloud-backed file system for the enterprise. In *Proceedings of the 10th USENIX conference on File and Storage Technologies*, pages 19–19. USENIX Association.

Waldman, M. and Mazieres, D. (2001). Tangler: a censorship-resistant publishing system based on document entanglements. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 126–135. ACM.

Waldman, M., Rubin, A. D., and Cranor, L. F. (2000). Publius: A robust, tamper-evident censorship-resistant web publishing system. In *9th USENIX Security Symposium*, pages 59–72.

Wang, D., Zhang, Q., and Liu, J. (2006). Partial network coding: Theory and application for continuous sensor data collection. In *Quality of Service, 2006. IWQoS 2006. 14th IEEE International Workshop on*, pages 93–101. IEEE.

Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D., and Maltzahn, C. (2006a). Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320. USENIX Association.

Weil, S. A., Brandt, S. A., Miller, E. L., and Maltzahn, C. (2006b). Crush: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 122. ACM.

Welch, B., Unangst, M., Abbasi, Z., Gibson, G. A., Mueller, B., Small, J., Zelenka, J., and Zhou, B. (2008). Scalable performance of the panasas parallel file system. In *FAST*, volume 8, pages 1–17.

Wilcox-O'Hearn, Z. and Warner, B. (2008). Tahoe: the least-authority filesystem. In *Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 21–26. ACM.

Wildani, A., Schwarz, T. J., Miller, E. L., and Long, D. D. (2009). Protecting against rare event failures in archival systems. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on*, pages 1–11. IEEE.

Wylie, J. J. (2011a). Finding the most fault-tolerant flat xor-based erasure codes for storage systems. In *Signals, Systems and Computers (ASILOMAR), 2011 Conference Record of the Forty Fifth Asilomar Conference on*, pages 1788–1792. IEEE.

Wylie, J. J. (2011b). List of most fault-tolerant flat xor-based erasure codes for storage systems. Technical report, HP Labs, Tech. Rep. HPL-2011-217.

Wylie, J. J., Bigrigg, M. W., Strunk, J. D., Ganger, G. R., Kiliccote, H., and Khosla, P. K. (2000). Survivable information storage systems. *Computer*, 33(8):61–68.

Wylie, J. J. and Swaminathan, R. (2007). Determining fault tolerance of xor-based erasure codes efficiently. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 206–215. IEEE.

Zhang, X., Neglia, G., Kurose, J., and Towsley, D. (2006). On the benefits of random linear coding for unicast applications in disruption tolerant networks. In *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2006 4th International Symposium on*, pages 1–7. IEEE.

Zhou, M., Zhang, R., Xie, W., Qian, W., and Zhou, A. (2010). Security and privacy in cloud computing: A survey. In *Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on*, pages 105–112. IEEE.

AUTHOR BIO

Verónica del Carmen Estrada Galiñanes joined the Complex Systems Group at the University of Neuchatel, Switzerland, in 2012, supervised by Prof. Pascal Felber. She holds a Master degree in Applied Computer Science from University of Tokyo, Japan, obtained in 2011, a graduate degree as Specialist in Cryptography and Security from the Higher Education Army Institute, Argentina obtained in 2005 and an Electronic Engineer degree from University of Buenos Aires, Argentina, obtained in 2003. Before she started building a scientific career, she worked several years in the private and the public sector.

Verónica's research interests include storage, security and distributed systems. In 2008, she received a Monbushogakusho scholarship from the Japanese MEXT research's agency. Her research also received support from Usenix, ACM and Google. In 2015, she received an SNF Doc Mobility grant to spend 6 months as a visitor scholar at the Storage Systems Research Center, University of California Santa Cruz to collaborate with Dr. Ethan Miller and other lab associated researchers. Her project was the design of next-generation erasure coding methods. She serves the community by volunteering at events and in projects that motivate and inspire students such as: ACM XRDS Crossroads, Rotary Club and Google Summer of Code.