2012

Improve the Performance and Scalability of RAID-6 Systems Using Erasure Codes

Chentao Wu
Virginia Commonwealth University

Improve the Performance and Scalability of RAID-6 Systems Using Erasure Codes

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

Chentao Wu
B. E., Huazhong Univ. of Sci. and Tech., Computer Sci. and Tech.
M. E., Huazhong Univ. of Sci. and Tech., Software Engineering

Director: Dr. Xubin He, Associate Professor
Department of Electrical and Computer Engineering

Virginia Commonwealth University
Richmond, Virginia
November 2012

Acknowledgements

I would like to thank my dissertation advisor Dr. Xubin He, for his constant support and guidance at VCU in these years. He not only teaches me academically and shares his profound knowledge on data storage systems, but also encourages me in my daily life as an older brother. I would like to thank Dr. Wei Zhang, Dr. Meng Yu, Dr. Robert H. Klenke and Dr. Preetam Ghosh for serving on my advisory committee.

The Storage Technology and Architecture Research (STAR) Lab at VCU have been very supportive in these years, but a special thank to Mr. Guanying Wu, Dr. Shenggang Wan, Dr. Huailiang Tan, Mr. Chao Yu and Mr. Tao Lyu for their continuous help and support.

I would also like to thank Prof. Changsheng Xie, Dr. Qiang Cao, Dr. Jianzhong Huang, Dr. Zhihu Tan, Dr. Jiguang Wan, etc., who gave me a lot support on both academic research and daily life at Wuhan National Lab for Optoelectronics (WNLO), Huazhong University of Science and Technology (HUST), Wuhan, China.

Finally, I would like to thank my parents and my younger brother for their understanding on my research. I would like to thank my wife Yizhe for her constant patience, love, and support in all of my endeavors for the last four years. Words cannot express my appreciation and gratefulness for finding you on this Earth and making the improbable a reality.

Contents

List of Tables ix

List of Figures xi

List of Algorithms xv

List of Symbols xvi

Preface xx

Abstract xxii

1 Introduction 1

1.1 Motivations 1

1.2 Problem Statement 3

1.3 Background and Related Work 3

1.3.1 Definitions 3

1.3.2 Existing Erasure Codes 5

1.3.3 Research on Performance in RAID Systems 9

1.3.4 Research on Scalability in RAID Systems 12

1.4 Research Approaches 16

2 H-Code **19**

2.1 Introduction 19

2.2 H-Code 23

2.2.1 Data/Parity Layout and Encoding of H-Code 23

2.2.2 Construction Process 25

2.2.3 Proof of Correctness 26

2.2.4 Reconstruction 29

2.3 Property Analysis 32

2.3.1 Optimal Property of H-Code 32

2.3.2 Partial Stripe Writes to Two Continuous Data Elements in H-Code . 34

2.3.3 Different Cases of Partial Stripe Writes to w Continuous Data Elements 35

2.4 Performance Evaluation 36

2.4.1 Evaluation Methodology 36

2.4.2 Numerical Results 40

2.5 Summary 45

3 HDP Code **47**

3.1 Introduction 47

3.2 Summary on Load Balancing of Various Parities and Our Motivation 50

3.2.1 Horizontal Parity (HP) 51

3.2.2 Diagonal/Anti-diagonal Parity (DP/ADP) 51

3.2.3 Vertical Parity (VP) 52

3.2.4 Motivation 52
3.3 HDP Code 53
3.3.1 Data/Parity Layout and Encoding of HDP Code 53
3.3.2 Construction Process 55
3.3.3 Proof of Correctness 55
3.3.4 Reconstruction Process 57
3.3.5 Property of HDP Code 59
3.4 Load Balancing Analysis 59
3.4.1 Evaluation Methodology 60
3.4.2 Numerical Results 63
3.5 Reliability Analysis 65
3.5.1 Fast Recovery on Single Disk 66
3.5.2 Parallel Recovery on Double Disks 67
3.6 Summary 68

4 Stripe-based Data Migration (SDM) Scheme 70

4.1 Introduction 70
4.2 Motivation 72
4.3 SDM Scheme 73
4.3.1 Priority Definition of Data/Parity Movements 74
4.3.2 Layout Comparison 75
4.3.3 Load Balancing Check in SDM 80
4.3.4 Data Migration 83
4.3.5 Data Addressing Algorithm 84

4.3.6 Property of SDM 84
4.4 Scalability Analysis 86
4.4.1 Evaluation Methodology 86
4.4.2 Numerical Results 89
4.4.3 Analysis 93
4.5 Summary 93

5 MDS Coding Scaling Framework (MDS-Frame) 95

5.1 Introduction 95
5.2 Motivation 96
5.3 MDS-Frame 97
5.4 Intermediate Code Layer 98
5.4.1 S_p-Code 99
5.4.2 S_{p-1}-Code and S_{p+1}-Code 104
5.4.3 Observations 106
5.5 Management Layer 107
5.5.1 Unified Management User Interface 107
5.5.2 Definition of the HE and LE Scaling 107
5.5.3 Scaling Algorithms 108
5.5.4 Overhead Analysis of Scaling between Intermediate Codes 109
5.5.5 Scaling Rules 110
5.6 Scalability Analysis 110
5.6.1 Evaluation Methodology 111
5.6.2 Numerical Results and Analysis 111

5.7 Summary 114

6 Conclusions **115**

Bibliography **117**

Vita **130**

List of Tables

2.1 50 Random Integer Numbers 38

2.2 Improvement of H-Code over Other Codes in terms of Average Partial Stripe Write Cost (Uniform Access) 41

2.3 Improvement of H-Code over Other Codes in terms of Average Partial Stripe Write Cost (Random Access) 43

2.4 Cost Of A Partial Stripe Write to w Continuous Data Elements in the Same Row ($p = 7$, Uniform Access) 44

3.1 Summary of Different Parities 53

3.2 50 Random Integer Numbers 62

3.3 Different L Value in Horizontal Codes 64

3.4 Reduced I/O of Single Disk failure in Different Codes 67

3.5 Recovery Time of Any Two Disk Failures in HDP Code ($p = 5$) 68

4.1 Summary on Various Fast Scaling Approaches 72

4.2 Priorities of Data/Parity Migration in SDM (For A Movement on Single Data/Parity Element) 74

4.3 Different overhead of Extended Disk(s) Labelling in SDM 77

4.4 Data Distribution of RAID-6 Scaling by Using SDM Scheme (RDP code, Scaling from 6 to 8 disks) 80
4.5 Data Distribution of RAID-6 Scaling by Using SDM Scheme (P-Code, Scaling from 6 to 7 disks) 80
4.6 Overhead of RAID-6 Scaling by Using SDM Scheme (RDP code, Scaling from 6 disks to 8 disks) 87
4.7 Overhead of RAID-6 Scaling by Using SDM Scheme (P-Code, Scaling from 6 disks to 7 disks) 88
4.8 Speed Up of SDM Scheme over Other RAID-6 Scaling Schemes in terms of Migration Time 93

5.1 Overhead of Typical scaling in MDS-Frame 109

List of Figures

1.1 EVENODD code for $p + 2$ disks when $p = 5$ and $n = 7$. 6

1.2 Blaum-Roth code for $p + 1$ disks ($p = 5$, $n = 6$). 6

1.3 Liberation code for $p + 2$ disks ($p = 5$, $n = 7$). 7

1.4 RDP code for $p + 1$ disks ($p = 5$, $n = 6$). 7

1.5 X-Code for p disks ($p = 5$, $n = 5$). 8

1.6 P-Code. 8

1.7 HoVer code for 6 disks ($n = 6$). 9

1.8 Reduce I/O cost when column 2 fails in RDP code ($p = 7$, $n = 8$). 11

1.9 Reduce I/O cost when column 2 fails in X-Code ($p = 7$, $n = 7$). 12

1.10 RAID-6 scaling in RDP from 6 to 8 disks using RR approach. 14

1.11 RAID-6 scaling in P-Code from 6 to 7 disks using Semi-RR approach. . . . 15

1.12 RAID-5 scaling from 4 to 5 disks using ALV approach (all data blocks are need to be migrated). 15

1.13 RAID-5 scaling from 4 to 5 disks using MDM approach. 15

1.14 RAID-0 scaling from 3 to 5 disks using FastScale approach. 16

2.1 Partial stripe write problem in RDP code for an 8-disk array ($p = 7$). 20

2.2 Partial stripe write problem in X-Code for a 7-disk array. 21

2.3 H-Code ($p = 7$). 24
2.4 Reconstruction Process of H-Code. 30
2.5 Partial stripe writes to two continuous data elements in H-Code for an 8-disk array. 35
2.6 Average number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 5$. . . . 41
2.7 Average number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 7$. . . . 42
2.8 Maximum number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 5$. . 43
2.9 Maximum number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 7$. . 44
2.10 Average number of I/O operations in column j of a partial stripe write to three continuous data elements of different codes in an 8-disk array ($p = 7, w = 3$). 45

3.1 Load balancing problem in RDP code for an 8-disk array. 49
3.2 Load balancing problem in P-Code for a 7-disk array. 50
3.3 HDP Code ($p = 7$). 54
3.4 Reconstruction by two recovery chains. 58
3.5 Write request distribution in *Microsoft Exchange* trace (most write requests are 4KB and 8KB). 60
3.6 Metric of load balancing L under various codes ($p = 5$). 63
3.7 Metric of load balancing L under various codes ($p = 7$). 64

3.8 Metric of load balancing L under uniform SW requests with different p in various horizontal codes. 65
3.9 Metric of load balancing L under uniform CW requests with different p in various horizontal codes. 65
3.10 Single disk recovery in HDP Code when column 0 fails. 66
3.11 Single disk recovery in HDP Code when column 0 fails. 67

4.1 Data movement of Block 0 in Figure 1.10. 75
4.2 RDP code for $p+1$ disks ($p=7$, $n=8$). 77
4.3 Data/parity migration in Layout Comparison Step (RDP code, scaling from 6 to 8 disks). 78
4.4 Data/parity migration in Layout Comparison Step (P-Code, scaling from 6 to 7 disks). 79
4.5 Data/parity migration in load balancing check step (RDP code, scaling from 6 to 8 disks). 82
4.6 Data/parity migration in load balancing check step (P-code, scaling from 6 to 7 disks). 83
4.7 I/O distribution in multiple stripes using SDM scheme. 89
4.8 Comparison on data distribution under various RAID-6 scaling approaches. 90
4.9 Comparison on data migration ratio under various RAID-6 scaling approaches. 90
4.10 Comparison on parity modification ratio under various RAID-6 scaling approaches. 91
4.11 Comparison on computation cost under various RAID-6 scaling approaches (The number of B XORs is normalized to 100%). 91

4.12 Comparison on total I/Os under various RAID-6 scaling approaches (The number of B I/O operations is normalized to 100%). 92

4.13 Comparison on migration time under various RAID-6 scaling approaches (The time $B * T_b$ is normalized to 100%). 92

5.1 MDS-Frame Architecture. 97

5.2 S-Codes ($p = 5$). 99

5.3 Reconstruction by two recovery chains in S_p-Code. 104

5.4 Total I/Os of various scaling processes in MDS-Frame vs. Round-Robin when $p = 5$ and $p = 7$ (The number of B I/O operations is normalized to 100%). 112

5.5 Computation cost of various scaling processes in MDS-Frame vs. Round-Robin when $p = 5$ and $p = 7$ (The number of B XORs is normalized to 100%). 112

5.6 Migration time of various scaling processes in MDS-Frame vs. Round-Robin when $p = 5$ and $p = 7$ (The time $B * T_e$ is normalized to 100%). . . 113

5.7 Migration time in scaling HDP→EVENODD under various p (the number of B I/O operations is normalized to 100%). 113

List of Algorithms

2.1 Reconstruction Algorithm of H-Code 31

3.1 Reconstruction Algorithm of HDP Code 59

4.1 Data Addressing Algorithm of RDP Scaling from n to $n+m$ disks Using SDM Scheme (where $n = p_1 + 1$, $n + m = p_2 + 1$, $p_1 < p_2$, p_1 and p_2 are prime numbers) 85

4.2 Data Addressing Algorithm of P-Code Scaling from n to $n+m$ disks Using SDM Scheme (where $n = p_1 - 1$, $n + m = p_2$, $p_1 \leq p_2$, p_1 and p_2 are prime numbers) 85

5.1 Add A New Code into MDS-Frame 107

5.2 Scaling Algorithm (S_{p-1}-Code$\rightarrow S_p$-Code) 109

List of Symbols

Symbols	Description
n	number of disks in a disk array (before scaling)
m	scaled number of disk(s) (m is negative when disk(s) are removed)
k	number of data disks in a disk array (before scaling)
k'	number of data disks in a disk array (after scaling)
p	a prime number
i, r	row ID (before scaling)
i'	row ID (after scaling)
j	column ID or disk ID (before scaling)
j'	column ID or disk ID (after scaling)
B	total number of data blocks (data elements)
$C_{i,j}$	element at the ith row and jth column
f_1, f_2 ($f_1 < f_2$)	two random failed columns with IDs f_1 and f_2
S	total number of stripes (before scaling)
S'	total number of stripes (after scaling)
S_{id}	stripe ID before scaling
S'_{id}	stripe ID after scaling

Symbols	Description
$\sum$	XOR operations between/among elements
(e.g., $\sum_{j=0}^{5} C_{i,j}$)	$(C_{i,0} \oplus C_{i,1} \oplus \cdots \oplus C_{i,5})$
$\langle\ \rangle$	modular arithmetic
(e.g., $\langle i \rangle_p$)	(i mod p)
w	number of continuous data elements in a partial stripe write
N_s	total access frequency of all stripe writes in the ideal sequence
$F(C_{i,j})$	access frequency of a partial stripe write with beginning data element $C_{i,j}$
$P(C_{i,j})$	access probability of a partial stripe write with beginning data element $C_{i,j}$
$S_w(C_{i,j})$	number of I/O operations caused by a partial stripe write to w continuous data elements with beginning element $C_{i,j}$
$S_{max.}(w)$	maximum number of I/O operations of a partial stripe write to w continuous data elements
$S_{avg.}(w)$	average number of I/O operations of a partial stripe write to w continuous data elements
$S_w^j(C_{i,j})$	number of I/O operation in column j of a partial stripe write to w continuous data elements with beginning element $C_{i,j}$
$S_{avg.}^j(w)$	average number of I/O operations in column j of a partial stripe write to w continuous data elements
$O_j(C_{i,j})$	number of I/O operations to column j, which is caused by a request to data element(s) with beginning element $C_{i,j}$

Symbols	Description
$O(j)$	total number of I/O operations of requests to column j in a stripe
O_{max}, O_{min}	maximum/minimum number of I/O operations among all columns
L	metric of load balancing in various codes
R_t	average time to recover a data/parity element
$N_{avg.}$	average number of elements can be recovered in a time interval R_t
P, Q	parity blocks (before scaling)
P', Q'	parity blocks (after scaling)
x, x_1, x_2	three random integers
n_s	number of stripes in each stripe set
n_e	total number of data elements in each stripe before scaling
n_{ec}	total number of data elements in each column per stripe set
R_d	data migration ratio
R_p	parity modification ratio
n_{io}	total number of I/O operations
T_b	access time of a read/write request to a block
T_m	migration time
w_s	word size

A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal Turing Machine.

Alan Mathison Turing

Preface

This dissertation is mainly based on the following papers which will be referred to in the text by their Roman numerals.

I **Chentao Wu**, Xubin He, Jizhong Han, Huailiang Tan, and Changsheng Xie. SDM: A Stripe-based Data Migration Scheme to Improve the Scalability of RAID-6. In: *Proceedings of the International Conference on Cluster Computing 2012 (Cluster 2012)*, September 24–28, 2012, Beijing, China.

II **Chentao Wu** and Xubin He. GSR: A Global Stripe-based Redistribution Approach to Accelerate RAID-5 Scaling. In: *Proceedings of the 41st International Conference on Parallel Processing (ICPP 2012)*, September 10–13, 2012, Pittsburgh, USA.

III **Chentao Wu**, Shenggang Wan, Xubin He, Guanying Wu, Shenggang Wan, Xiaohua Liu, Qiang Cao, and Changsheng Xie. HDP Code: A Horizontal-Diagonal Parity Code to Optimize I/O Load Balancing in RAID-6. In: *Proceedings of the 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2011)*, June 27–30, 2011, Hong Kong, China.

IV **Chentao Wu**, Shenggang Wan, Xubin He, Qiang Cao, and Changsheng Xie. H-Code: A Hybrid MDS Array Code to Optimize Partial Stripe Writes in RAID-6.

In: *Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2011)*, May 13–16, 2011, Anchorage, USA.

V **Chentao Wu** and Xubin He. A Flexible Framework to Enhance RAID-6 Scalability via Exploiting the Similarities among MDS Codes. Under review.

Abstract

IMPROVE THE PERFORMANCE AND SCALABILITY OF RAID-6 SYSTEMS USING ERASURE CODES

By Chentao Wu

A Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2012

Major Director: Dr. Xubin He, Associate Professor, Department of Electrical and Computer Engineering

RAID-6 is widely used to tolerate concurrent failures of any two disks to provide a higher level of reliability with the support of erasure codes. Among many implementations, one class of codes called **M**aximum **D**istance **S**eparable (**MDS**) codes aims to offer data protection against disk failures with optimal storage efficiency. Typical MDS codes contain horizontal and vertical codes.

However, because of the limitation of horizontal parity or diagonal/anti-diagonal parities used in MDS codes, existing RAID-6 systems suffer several important problems on performance and scalability, such as low write performance, unbalanced I/O, and high migration cost in the scaling process.

To address these problems, in this dissertation, we design techniques for high performance and scalable RAID-6 systems. It includes high performance and load balancing erasure codes (H-Code and HDP Code), and Stripe-based Data Migration (SDM) scheme. We also propose a flexible MDS Scaling Framework (MDS-Frame), which can integrate H-Code, HDP Code and SDM scheme together. Detailed evaluation results are also given in this dissertation.

Key Words: RAID-6; MDS Codes; Horizontal Parity; Diagonal/Anti-diagonal Parity; Performance Evaluation; Load Balancing; Scalability

Chapter 1

Introduction

1.1 Motivations

Redundant **A**rrays of **I**nexpensive (or **I**ndependent) **D**isks (**RAID**) [57, 16] is an efficient approach to supply high reliability and high performance storage services with acceptable spatial and monetary cost. In recent years, RAID-6 [73] has received much attention because it can tolerate concurrent failures of any two disks. It has been shown to be of increasing importance due to technology trends [18, 63] and the fact that the possibility of concurrent disk failures increases [72, 58] as the system scale grows.

Many implementations of RAID-6 based on various erasure coding technologies, which aim to offer data protection against disk failures with given amount of redundancy. They play a significant role in disk arrays, many companies and major technology corporations are performing active research in this area, such as Allmydata [1], Cleversafe [17, 68], Data Domain [91], Network Appliance [3], Panasas [4, 79], Hewlett Packard [83, 31], IBM [34, 35, 33, 48, 9], Microsoft [41, 42, 38, 39, 54, 49, 40], etc. Several academic projects

also focus on this aspect, such as LoCI [7], Oceanstore [50, 69], Pergamum [74], RAIF [47], RAIN [12], and so on.

Maximum **D**istance **S**eparable (**MDS**) codes [67, 11, 8, 18, 10, 61, 85, 86, 45, 24, 15] are optimal erasure codes, which are the most popular erasure codes and offer optimal storage efficiency. Typical MDS codes can be further categorized into horizontal codes [67, 11, 8, 18, 10, 61] and vertical codes [85, 86, 45, 15].

However, with the increasing requirements by users, RAID-6 faces several significant problems on performance and scalability. These problems includes,

1. Poor write performance. Compared to RAID-0 or RAID-5, RAID-6 has much lower write performance, especially for single write and partial stripe write [19, 51, 46, 13, 30].

2. Unbalanced I/O distribution. All existing horizontal codes [67, 11, 8, 18, 10, 61] and some vertical codes [45] have dedicated parity or data disks, and suffer unbalanced I/Os in these disks. Unbalanced I/Os can significantly decrease the performance of a RAID system [26, 71, 44, 5, 36, 6].

3. Difficult to scale efficiently. Fast scaling is a significant aspect in cloud computing [2]. Existing scaling approaches [29, 55, 88, 28, 89, 37, 90] are designed for RAID-0 or RAID-5, which cannot adapt various MDS coding methods in RAID-6. They cause high migration and computation cost, thus the scaling process is very slow.

4. Poor flexibility [59]. Existing MDS codes [67, 11, 8, 18, 10, 61, 15, 86, 45] have no relationship between each other. It is difficult to combine these codes into a general purpose framework.

In summary, existing approaches and solutions are insufficient to support both high performance and high scalability for RAID-6 systems, which motivates us to design some new techniques to improve the performance and scalability.

1.2 Problem Statement

In this dissertation, we state our research purpose as: *Improve the performance and scalability of RAID-6 systems by investigating erasure codes. Our design provides high write performance and I/O load balancing. Our design also achieves high scalability by stripe-based data migration schemes. Finally we integrate our techniques into a flexible framework to support various RAID-6 codes.*

1.3 Background and Related Work

To improve the efficiency, performance, reliability and scalability of the RAID-6 storage systems, different MDS coding approaches are proposed. In this section we discuss the existing MDS codes and the related works on performance and scalability issues in RAID systems.

1.3.1 Definitions

Prior to the introduction on the background and related works, the following definitions are given for RAID-6 systems according to previous literatures [34, 77],

- ***Element:*** The element is the fundamental unit for constructing codes. An element can be a data bit up to several data blocks. There are two types of elements, data elements and parity elements.

- ***Stripe:*** A stripe is a complete (connected) set of data and parity elements that are dependently related by parity computation relations [34]. Typically, a stripe is an element matrix in RAID-6.

- ***Stripe Size:*** Total number of data and parity elements in a stripe.

- ***Parity Chain:*** A parity chain includes a parity element and all data elements used to encode it.

- ***Parity Chain Length:*** Total number of data and parity elements in a parity chain.

- ***Horizontal Parity Chain (or Row Parity Chain):*** A parity chain with all elements shares a same row. The corresponding parity element is called "**horizontal parity element**".

- ***Vertical Parity Chain:*** A parity chain with all elements distributes in multiple rows. The corresponding parity element is "**vertical parity element**".

- ***Diagonal/Anti-diagonal Parity Chain (two special types of vertical parity chains):*** A parity chain with all data elements follows diagonal/anti-diagonal distribution. The corresponding parity element is "**diagonal/anti-diagonal parity element**".

- ***Horizontal Code:*** An MDS code includes horizontal parity chains, and all parity elements are in dedicated parity columns.

- ***Vertical Code:*** An MDS code which is not a horizontal code.

1.3.2 Existing Erasure Codes

Many RAID-6 implementations are presented based on various erasure coding technologies, which are divided into two categories: MDS codes and non-MDS codes.

Maximum **D**istance **S**eparable (**MDS**) codes in RAID-6 can be further divided into two subclasses: horizontal codes and vertical codes. Horizontal codes include Reed-Solomon codes [67], Cauchy Reed-Solomon codes [11], EVENODD code [8], RDP code [18], Blaum-Roth code [10], Liberation code [61] and Liber8tion code [60]. Vertical codes contain B-Code [85], X-Code [86], P-Code [45] and Cyclic code [15]. Typically, non-MDS codes involve LDPC codes [25, 53, 64], WEAVER code [33], HoVer codes [34], MEL code [83], Pyramid code [38], Flat XOR-Code [31], Code-M [77] and Partial-MDS Codes [9]. Besides these codes in RAID-6, other erasure codes such as RSL-Code [21], RL-Code [22], STAR code [41, 42, 54], HoVer codes [34] and GRID Codes [52], which can tolerate concurrent failures of three or more disks. Recently, research on erasure codes turns to the applications in cloud computing [49, 40].

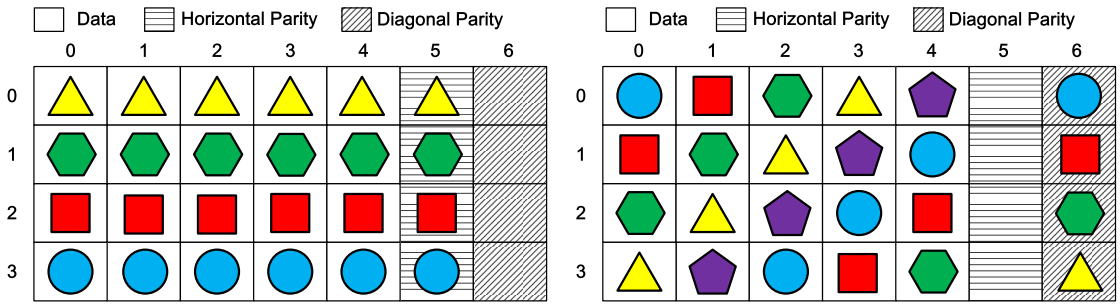
Here we briefly overview several popular MDS and non-MDS codes.

Reed-Solomon codes [67, 32] are based on addition and multiply operations over *Galois Field* arithmetic (GF(2^{w_s})), where w_s is the word size and $2^{w_s} - 1$ is the maximum number of disks in a disk array. And an addition operation is equivalent to an XOR operation, while the multiplication is implemented with high complexity and high overhead.

Cauchy Reed-Solomon codes [11] are described by Binary Distribution Matrix (BDM) originally [61], which are variants of Reed-Solomon codes by using Cauchy matrix. They change multiply operations of Reed-Solomon codes to additional XOR operations, which

reduce partial cost of encoding and decoding but also have large amount of XOR operations [65].

EVENODD code [8] is shown in Figure 1.1, which is a typical horizontal code and consists of horizontal and diagonal parity chains. It performs lower cost on encoding/decoding than all variants of Reed-Solomon codes.

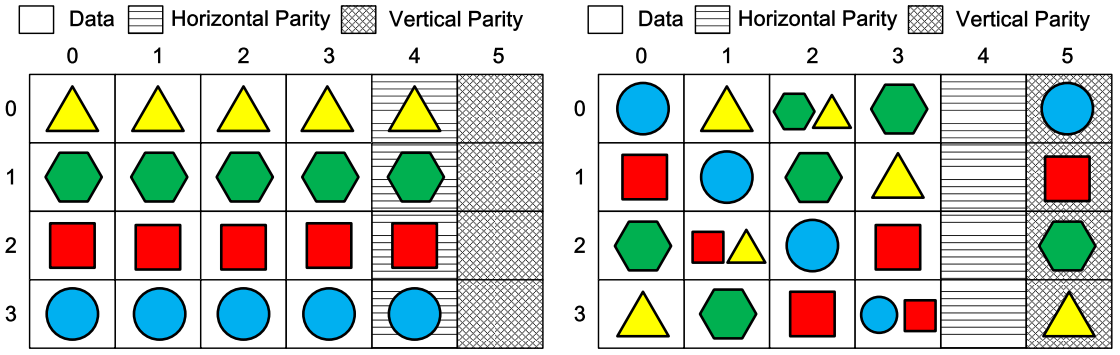


(a) Horizontal parity coding. (b) Diagonal parity coding.

Figure 1.1: EVENODD code for $p + 2$ disks when $p = 5$ and $n = 7$.

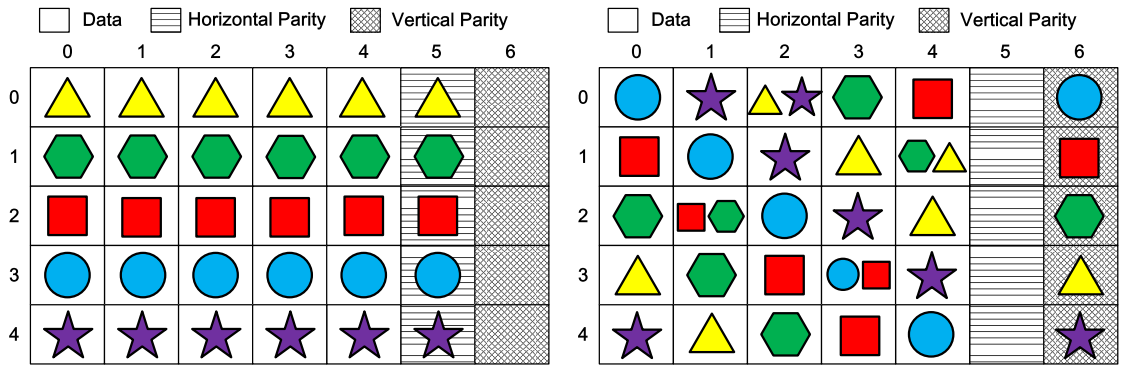
$S = C_{0,4} \oplus C_{1,3} \oplus C_{2,2} \oplus C_{3,1}$ and participates in the encoding of all diagonal parities.

Blaum-Roth, Liberation and Liber8tion codes (shown in Figures 1.2 and 1.3) [10, 61, 60, 62] are three separate "*minimum density codes*" [62], which provides near-optimal encoding/decoding performance and low overhead on updating single data element.



(a) Horizontal parity coding. (b) Vertical parity coding.

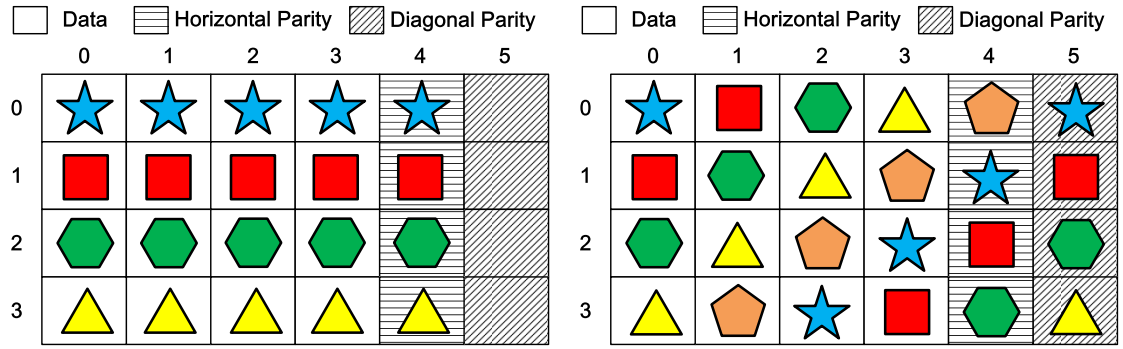
Figure 1.2: Blaum-Roth code for $p + 1$ disks ($p = 5$, $n = 6$).



(a) Horizontal parity coding. (b) Vertical parity coding.

Figure 1.3: Liberation code for $p + 2$ disks ($p = 5$, $n = 7$).

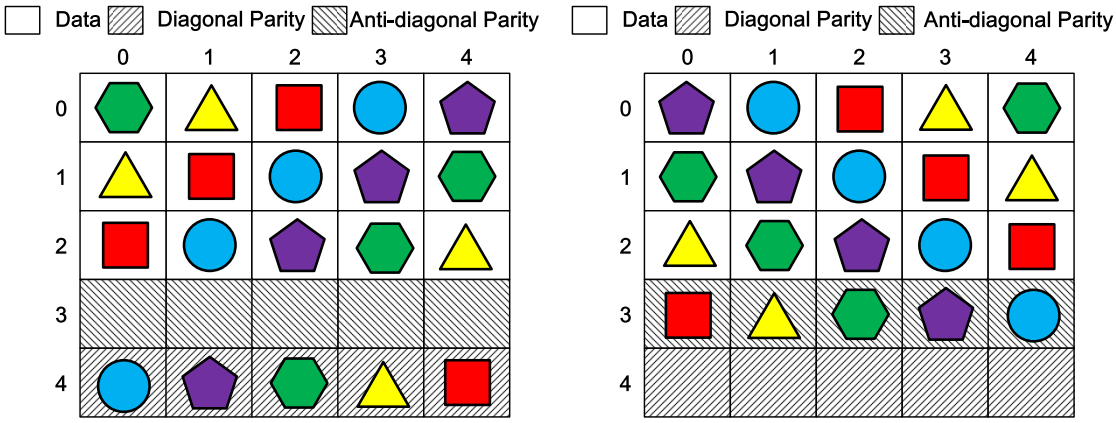
RDP code [18] is shown in Figure 1.4, which retains the horizontal parity chains as EVENODD while changes the diagonal layout. The special diagonal chain of RDP is that most horizontal parities take part in the construction of diagonal parities, which can decrease the reconstruction overhead.



(a) Horizontal parity coding. (b) Diagonal parity coding.

Figure 1.4: RDP code for $p + 1$ disks ($p = 5$, $n = 6$).

X-Code [86, 75] (shown in Figure 1.5) is a classic vertical code and the layout of X-Code is made up by diagonal and anti-diagonal parity chains. It has optimal computational complexity and optimal update complexity.

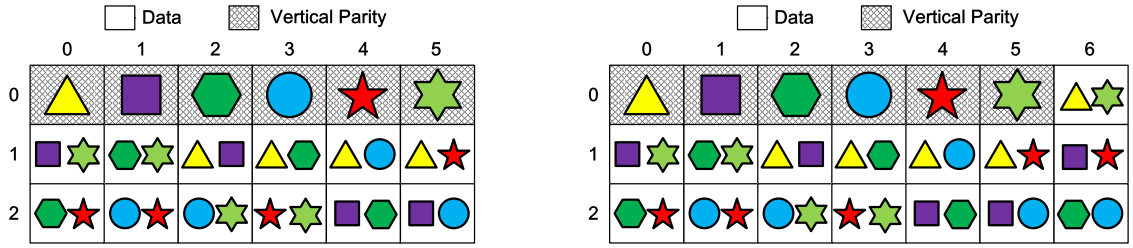


(a) Diagonal parity coding.

(b) Anti-diagonal parity coding.

Figure 1.5: X-Code for p disks ($p = 5$, $n = 5$).

P-Code [45] (shown in Figure 1.6) is another type of vertical code as Cyclic [15]. The stripe size of P-Code is $\frac{p-1}{2} * n$, where n is the total number of disks in a disk array. P-Code has many properties, such as optimal encoding/decoding computation complexity and optimal update complexity.

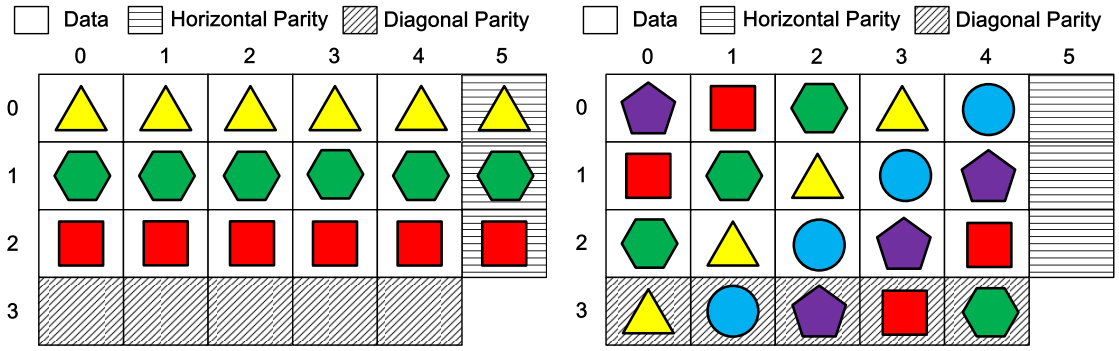


(a) Vertical parity coding for $p - 1$ disks ($p = 7$, $n = 6$).

(b) Vertical parity coding for p disks ($p = 7$, $n = 7$).

Figure 1.6: P-Code.

HoVer codes [34] (shown in Figure 1.7) is a family of XOR-based erasure codes, which can tolerate concurrent failures of two or more disks and provide flexible on implementation by adjusting parameters.



(a) Horizontal parity coding. (b) Diagonal parity coding.

Figure 1.7: HoVer code for 6 disks ($n = 6$).

Code-M [77] is also a minimal density code, which gives another way to construct a parity chain by crossing multiple stripes. It can sharply reduce the recovery time compared to RDP.

1.3.3 Research on Performance in RAID Systems

Stripe Write Performance

From 1990s, several methods [19, 51, 46, 13, 30] are proposed to improve the stripe write performance of disk arrays. In 1994, David *et al.* [19] find a way to reduce the penalty of a partial stripe write. Later, some methods [51, 46] are appeared to improve the write performance of RAID-5 systems. Recently, high partial stripe write performance are desired [13, 30] in higher-level software solution for RAID systems. Typically, these methods are based on RAID-5, which may not have positive effects on the complex layout of RAID-6.

Load Balancing Approaches

Research on dynamic load balancing starts at early 1990s. In 1993, Ganger *et al.* [26] discuss two load balancing approaches, disk striping and conventional data placement. Later, several dynamic load balancing approaches are proposed [71, 44, 5, 36, 6]. They can adjust the unbalanced I/O in data disks according to various access patterns of different applications or workloads. Typically, these approaches in disk array cost some resources to monitor the status of various storage devices and find the disks with high or low workload, then do some dynamic adjustment according to the monitoring results. While in RAID-6 storage system, it is hard to transfer the high workload in parity disks to any other disks, which breaks the construction of erasure code thus damages the reliability of the whole storage system. On the other hand, typical load balancing approaches bring additional overhead to the disk array.

Industrial products based on RAID-6 like EMC CLARiiON [20], which uses static parity placement to provide load balancing in each eight stripes, but it also has additional overhead to handle the data placement and still suffers unbalanced I/O in each stripe.

Disk Failure Recovery Performance

In recent years, research on disk failure recovery becomes a hot topic. Tian *et al.* [76] propose a novel dynamic data reconstruction optimization algorithm, which is based on the access locality and the frequently accessed areas have the higher priority to be recovered. It can accelerate the recovery process in terms of user response time and reconstruction time. Workout [82] significantly boost RAID reconstruction performance via outsourcing popular read/write requests originally targeted at the degraded RAID set to a surrogate RAID set during reconstruction. In 2010, Xiang *et al.* [84] give a hybrid recovery approach

called RDOR, which sharply decreases the recovery I/O cost by using both horizontal and diagonal parities to recover single disk failure. Zhu *et al.* [92, 93] inherit the work of RDOR and propose a heterogeneous recovery scheme for various RAID-6 codes. VDF [78] improve the recovery efficiency by an asymmetric cache policy. Here RDOR approach is briefly introduced and will be discussed in Chapter 3.

RDOR [84] is a hybrid recovery approach by using both horizontal and diagonal parities to recover single disk failure in RDP. It can minimize I/O cost and has well balanced I/O in other disks (except the failed disk). For example, as shown in Figure 1.8, data elements A, B and C are recovered by their diagonal parity while D, E and F are recovered by their horizontal parity. By this method, some elements (e.g., $C_{3,0}$) can be shared to recover another parity, which can reduce the number of read operations. Actually, up to 22.60% disk access time and 12.60% recovery time are decreased in the simulation [84].

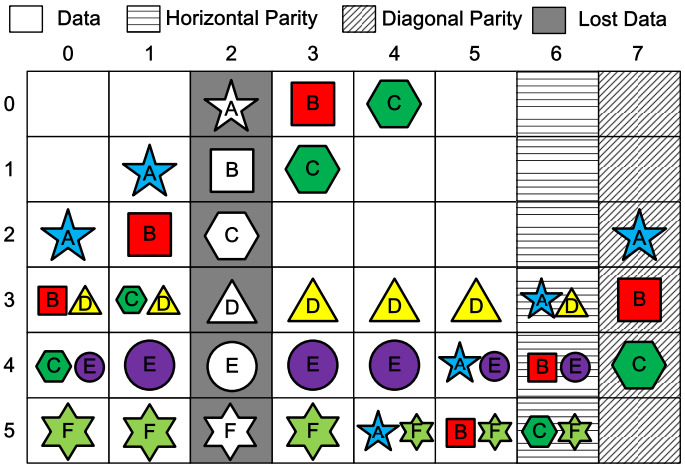


Figure 1.8: Reduce I/O cost when column 2 fails in RDP code ($p = 7$, $n = 8$).

However, this approach has less effect on vertical codes like X-Code. As shown in Figure 1.9, data elements A, B, C and F are recovered by their anti-diagonal parity while D, E and G are recovered by their diagonal parity. Though X-Code recovers more elements

compared to RDP in these examples, but X-Code share fewer elements than RDP thus has less effects to reduce the I/O cost when single disk fails.

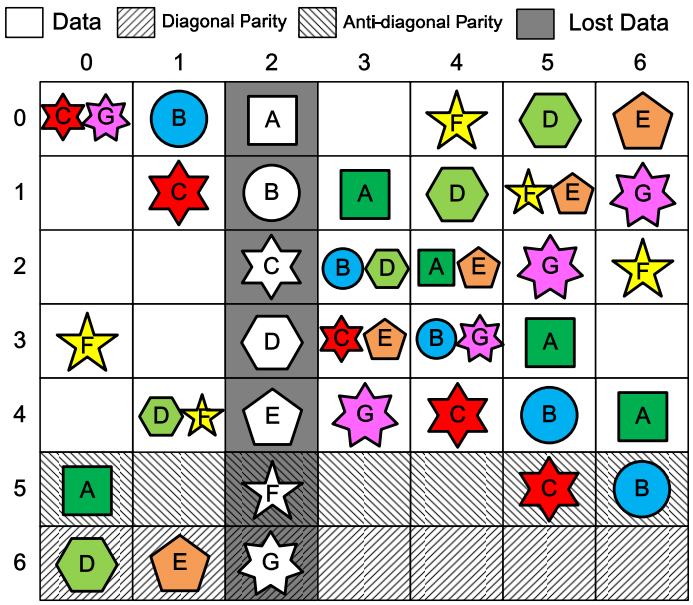


Figure 1.9: Reduce I/O cost when column 2 fails in X-Code ($p = 7$, $n = 7$).

RDOR is an efficient way to recover single disk failure, but it cannot reduce I/O cost when parity disk fails. For example, as shown in Figure 1.8, when column 7 fails, nearly all data should be read and the total I/O cost cannot be reduced.

1.3.4 Research on Scalability in RAID Systems

Desired Scaling Features in RAID-6

To extend a disk array, some data need to be migrated to the new disk(s) to achieve a balanced data distribution. During data migration, we prefer to keep an evenly distributed workload and minimize the data/parity movement. Combined with existing scaling approaches [90] and the real cases in RAID-6, the following four features are typically desired,

- Feature 1 (*Uniform Data Distribution*): Each disk has the same amount of data blocks to maintain an even workload.

- Feature 2 (*Minimal Data & Parity Migration*): By scaling m disks to a RAID-6 system with k data disks storing B data blocks, the expected total number of data movements is $m * B/(m+k)$ (scale-up, extending disks) or $|m| * B/n$ (scale-down, removing disks).

- Feature 3 (*Fast Data Addressing*): The locations of blocks in the array can be efficiently computed.

- Feature 4 (*Minimal Parity Computation & Modification*): A movement on data block could bring modification cost on its corresponding original parities and computation cost on new parities, so movements on data blocks should be limited in the original parity chain and thus parity blocks can be retained without any change.

Existing Fast Scaling Approaches

Existing approaches to improve the scalability of RAID systems include Round-Robin (RR) [29, 55, 88], Semi-RR [28], ALV [89], MDM [37], FastScale [90], etc. To clearly illustrate various strategies in RAID-6, we use P/Q (e.g., P_1 and Q_1) to delegate various parity blocks before scaling and P'/Q' (e.g., P'_1 and Q'_1) for the parity blocks after scaling. If the parity block is still presented by P/Q after scaling, it means that parity is unchanged.

Traditional RR and Semi-RR approaches can be used in RAID-6 under two restrictions. First, all data blocks are migrated based on round-robin order in the scaling process. Second, all parity blocks are retained without any movement.

For a traditional RR scaling approach (as shown in Figure 1.10), obviously, all data blocks are migrated and all parities need to be modified after data migration. Although RR is a simple approach to implement on RAID-6, it brings high overhead.

Based on RR approach, Brown [55] designed a reshape toolkit in the Linux kernel (MD-Reshape), which writes mapping metadata with a fixed-size window. Due to the limitation of RR approach, metadata are frequently updated by calling MD-Reshape function, which is inefficient.

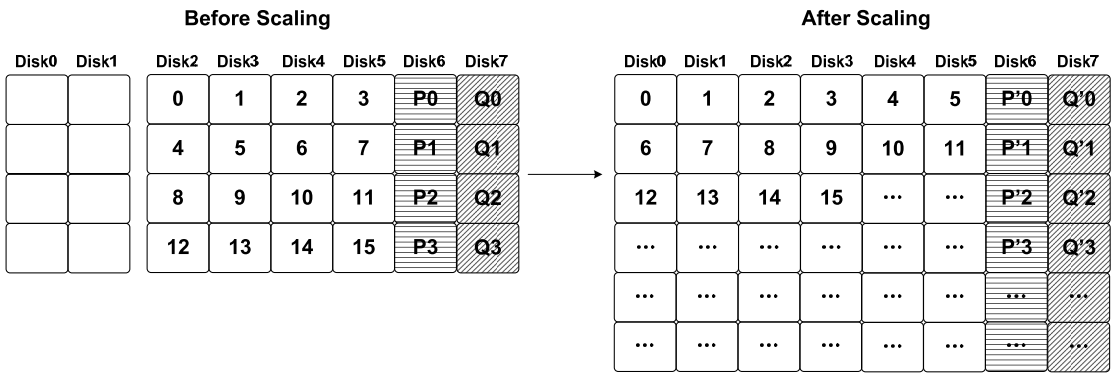


Figure 1.10: RAID-6 scaling in RDP from 6 to 8 disks using RR approach.

Semi-RR [28] is proposed to decrease high migration cost in RR scaling, shown in Figure 1.11. Unfortunately, by extending multiple disks, the data distribution is not uniform after scaling [28].

ALV [89] and MDM [37] are RAID-5 scaling approaches, which are shown in Figures 1.12 and 1.13, respectively. They cannot be applied in RAID-6. Different from RR-based approaches, ALV changes the movement order of migrated data and aggregate these small I/Os, which can improve the migration efficiency and updates of metadata. MDM eliminates the parity modification/computation cost and decreases the migration cost.

As shown in Figure 1.14, FastScale [90] takes advantages of both RR and Semi-RR approaches, which maintains a uniform data distribution, minimal data migration and

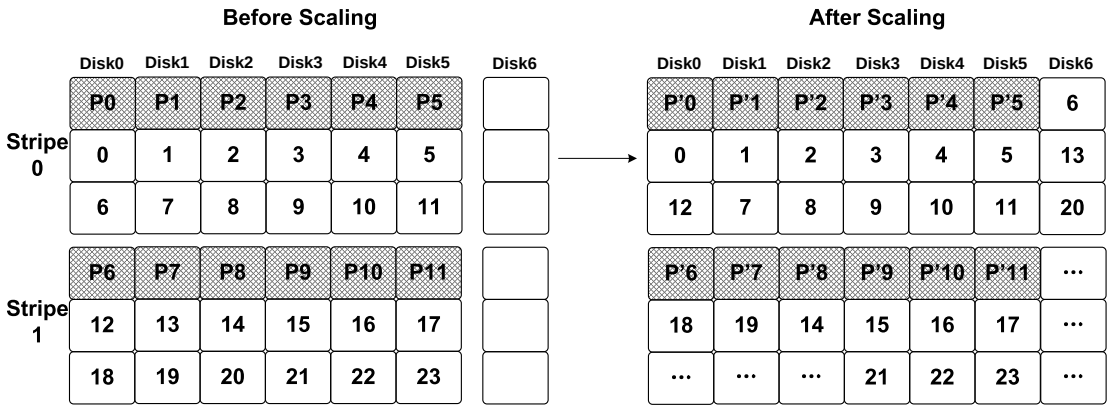


Figure 1.11: RAID-6 scaling in P-Code from 6 to 7 disks using Semi-RR approach.

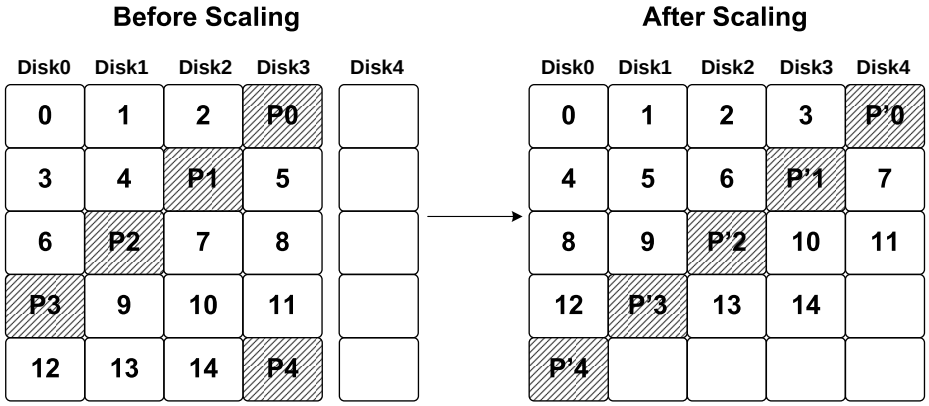


Figure 1.12: RAID-5 scaling from 4 to 5 disks using ALV approach (all data blocks are need to be migrated).

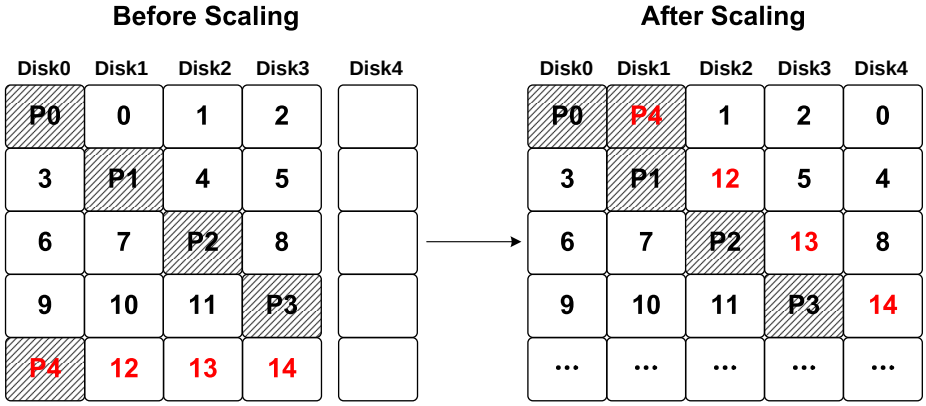


Figure 1.13: RAID-5 scaling from 4 to 5 disks using MDM approach.

fast data addressing. However, it only focuses on RAID-0 and doesn't support parity movement.

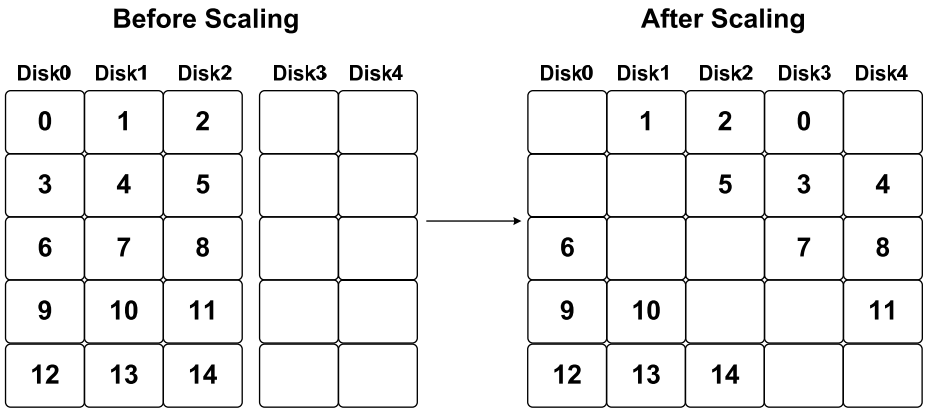


Figure 1.14: RAID-0 scaling from 3 to 5 disks using FastScale approach.

Except for the above scaling approaches, there are also some RAID-based systems which focus on the scalability issue. In 1990s, HP AutoRAID [81] permits an online expansion of disk array. Later, several RAID-based architectures [43, 70] are proposed for large scale storage systems, and scalability is one of the most significant impacts in these systems. Brinkmann *et al.* [14] gives mathematical analysis on a storage system by adding several disks. Franklin *et al.* [23] introduces a good way to support extension of RAID systems, but it need an additional disk as spare space. Recently, by the support of different file systems, RAID-Z [13] and HDFS RAID [80] achieves acceptable scalability in distributed storage systems.

1.4 Research Approaches

In this dissertation, we design the following techniques for high performance and scalable RAID-6 systems:

To improve the partial stripe write performance, we propose a new XOR-based MDS array code, named **Hybrid Code (H-Code)**, which optimizes partial stripe writes for RAID-6 by taking advantages of both horizontal and vertical codes. H-Code is a solution for an array of $(p+1)$ disks, where p is a prime number. Unlike other codes, the horizontal parity of H-Code ensures a partial stripe write to continuous data elements in a row share the same row parity chain, which can achieve optimal partial stripe write performance. Not only within a row but also within a stripe, H-Code offers optimal partial stripe write complexity to two continuous data elements and optimal partial stripe write performance among all MDS codes to the best of our knowledge.

To balance the I/O distribution in RAID-6 systems, we propose a new parity called **Horizontal-Diagonal Parity (HDP)**, which takes advantages of both horizontal and diagonal/anti-diagonal parities. The corresponding MDS code, called HDP code, distributes parity elements uniformly in each disk to balance the I/O workloads. HDP also achieves high reliability via speeding up the recovery under single or double disk failure.

To improve the scalability of RAID-6, we design a novel **Stripe-based Data Migration (SDM)** scheme for large scale storage systems based on RAID-6. SDM is a stripe-level scheme, and the basic idea of SDM is optimizing data movements according to the future parity layout, which minimizes the overhead of data migration and parity modification. SDM scheme also provides uniform data distribution, fast data addressing and migration.

To integrate these approaches into a RAID-6 system, we propose a novel **MDS** Code Scaling **Frame**work (**MDS-Frame**), which is a unified management scheme on various MDS codes to achieve high scalability. MDS-Frame bridges various MDS codes to achieve flexible scaling via several intermediate codes.

The rest of the dissertation is organized as follows. We discuss H-Code and HDP Code in Chapters 2 and 3. Chapter 4 describes how to accelerate RAID-6 scaling by using SDM scheme. We explain MDS-Frame in Chapter 5. Finally we conclude this dissertation in Chapter 6.

Chapter 2

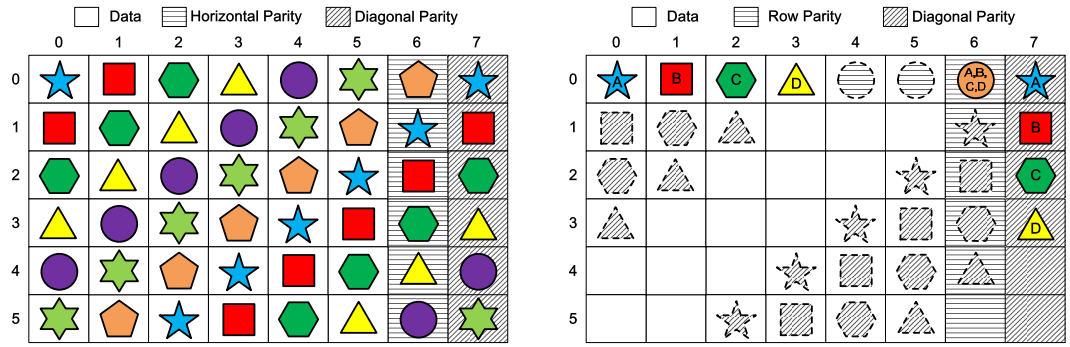
H-Code

2.1 Introduction

As introduced in Chapter 1, there are many implementations of RAID-6 based on various of erasure coding technologies, one class of codes called **M**aximum **D**istance **S**eparable (**MDS**) codes [67, 11, 8, 18, 10, 61, 15, 86, 45, 24]. They offer optimal ***full stripe write*** (a new write or an update) complexity, but the complexity of ***partial stripe write*** and single write (also called "short write", a partial stripe write to single element) is not satisfied by storage system designers [19, 51, 46, 13, 30].

A typical RAID-6 storage system (based on horizontal codes) is composed of $k + 2$ disk drives. The first k disk drives are used to store original data, and the last two, named P and Q, are used as parity disk drives. Due to the P parity, in the case of partial stripe write in a row, horizontal codes might get less I/O operations in most time, but suffer from unbalanced I/O distribution. Let's take an example of RDP codes [18] whose diagonal parity layout is shown in Figure 2.1(a). If there is a partial stripe write to 4 continuous elements ("continuous" means logically continuous among the disk arrays in

encoding/decoding) in a row as shown in Figure 2.1(b), it results in 4 reads and 4 writes to the Q parity disk (disk 7), but one read and one write to other disks (disk 0, 1, 2, 3 and 6). As the system scale grows, this problem cannot be resolved by shifting the stripes' parity strips among all the disks just as RAID-5. In addition, horizontal codes also have limitation on high single write complexity. Blaum *et al.* have proved that with a i-row-j-column matrix of data elements, at least $(i * j + j - 1)$ data elements participate in the generation of Q parities [10]. Thus the cost of a single block write in horizontal codes requires more than two additional writes on average, which is the lower bound of a theoretically ideal RAID-6 codes.



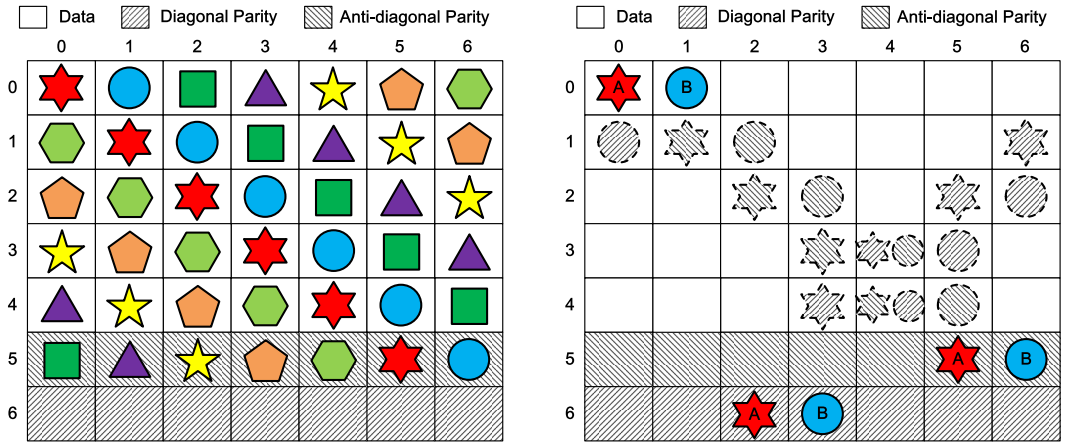
(a) Diagonal parity layout of RDP Code with $p + 1$ disks ($p = 7$): a diagonal element can be calculated by XOR operations among the corresponding elements, e.g., $C_{0,7} = C_{0,0} \oplus C_{5,2} \oplus C_{4,3} \oplus C_{3,4} \oplus C_{2,5} \oplus C_{1,6}$.

(b) A partial stripe write to 4 continuous data elements: A, B, C and D. The I/O operations in disks 0, 1, 2, 3 and 6 are all 1 read and 1 write, while in disk 7 are 4 reads and 4 writes. It shows that the workload in Disk 7 is very high, which may lead to a sharp decrease of reliability and performance of the system).

Figure 2.1: Partial stripe write problem in RDP code for an 8-disk array ($p = 7$).

Vertical codes, such as X-Code [86], Cyclic code [15], and P-Code [45], typically offer good single write complexity and encoding/decoding computational complexity as well as high storage efficiency. With special data/parity layout, they do not adopt row parity. In vertical codes, partial stripe write to multiple data elements in a row involves the generation of different parity elements. Let's take an example of X-Code whose anti-diagonal parity

layout is shown in Figure 2.2(a). Consider the following scenario as described in Figure 2.2(b). There is a partial stripe write to two data elements in the first row of an X-Code based 7-disk RAID-6 matrix. We notice that the two elements are in four different parity chains. Therefore, this partial stripe write results in $(2+4)$ reads and $(2+4)$ writes, for a total of 12 I/O operations. If the codes adopt row parity, the two elements might be only in three different parity chains. It can result in $(2+3)$ reads and $(2+3)$ writes, or 10 I/O operations in total. In other words, it reduces two I/O operations by adopting row parity. We evaluate the partial stripe write complexity in general and only focus on the short partial stripe writes, which affect no more than $(n-3)$ data elements for an array of n disks. Consider the following case, a partial stripe write to w continuous data elements in v parity chains. Typically, it results in one read and one write to each data element, and one read and one write to each parity element. Therefore, the number of disk I/O operations (denoted by S_w) is,



(a) Anti-diagonal parity layout of X-Code with p disks ($p = 7$): an anti-diagonal element can be calculated by XOR operations among the corresponding data elements, e.g., $C_{5,0} = C_{0,2} \oplus C_{1,3} \oplus C_{2,4} \oplus C_{3,5} \oplus C_{4,6}$.

(b) A partial stripe write to two continuous data elements: A and B. The I/O operations are $(2+4)$ reads and $(2+4)$ writes.

Figure 2.2: Partial stripe write problem in X-Code for a 7-disk array.

$$S_w = 2 * (w + v) \tag{2.1}$$

When the number of data elements affected by a partial stripe write in a row increases, the disk I/O reduced by adopting a row parity will increase. Similar problem also exists in Cyclic Code [15] and P-code [45]. However, almost all the horizontal codes adopt row parity, which might cause less partial stripe write cost. Jin *et al.* [45] mentioned a tweaked RDP code, which is a semi-vertical code composed of row parity and diagonal parity. However, for its individual diagonal parity disk, it still suffers from the unbalanced I/O distribution caused by partial stripe write to multiple data elements in a row as same as the horizontal codes.

To address these problems, we propose a novel XOR-based RAID-6 code, named **Hybrid Code (H-Code)**, which takes advantages of both horizontal and vertical codes. The parities in H-Code are classical row parity and anti-diagonal parity. H-code does not have a dedicated anti-diagonal parity strip, while it distributes the anti-diagonal parity elements among disks in the array. Its horizontal parity makes sure a partial stripe write to continuous data elements in a row share the same row parity chain, which achieves optimal partial stripe write performance. Depending on the number of disks in an MDS array, we design H-Code, which is a solution for n disks ($n = p + 1$), where p is a prime number.

The rest of the chapter is organized as follows. The design of H-Code is described in detail in Section 2.2. Property analysis and evaluation are given in Section 2.3 and Section 2.4. And the summary of this chapter is presented 2.5.

2.2 H-Code

To overcome the shortcomings of vertical and horizontal MDS codes, we present a hybrid MDS code scheme, named H-Code, to take advantage of both vertical and horizontal codes and is a solution for n disks ($n = p + 1$), where p is a prime number.

2.2.1 Data/Parity Layout and Encoding of H-Code

H-Code is represented by a $(p-1)$-row-$(p+1)$-column matrix with a total of $(p-1)*(p+1)$ elements. There are three types of elements in the matrix: ***data elements***, ***horizontal parity elements***, and ***anti-diagonal parity elements***. Assume $C_{i,j}$ ($0 \leq i \leq p-2$, $0 \leq j \leq p$) represents the element at the ith row and the jth column. The last column (column p) is used for horizontal parity. Excluding the first (column 0) and the last (column p) columns, the remaining matrix is a $(p-1)$-row-$(p-1)$-column square matrix. H-Code uses the anti-diagonal part of this square matrix to represent anti-diagonal parity.

Horizontal parity and anti-diagonal parity elements of H-Code are constructed based on the following encoding equations.

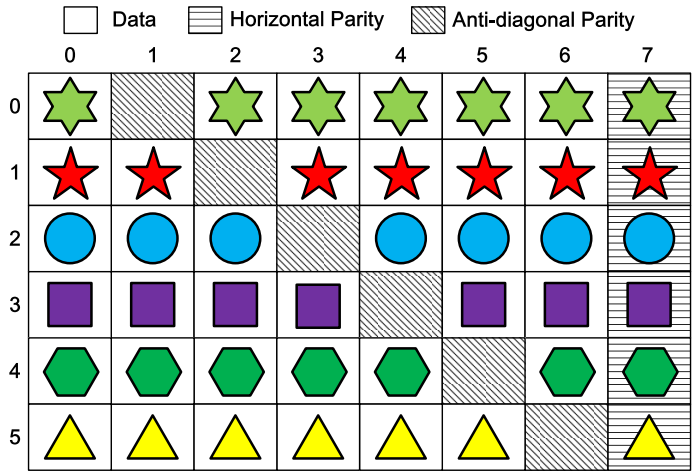
Horizontal parity:

$$C_{i,p} = \sum_{j=0}^{p-1} C_{i,j} \quad (j \neq i+1) \tag{2.2}$$

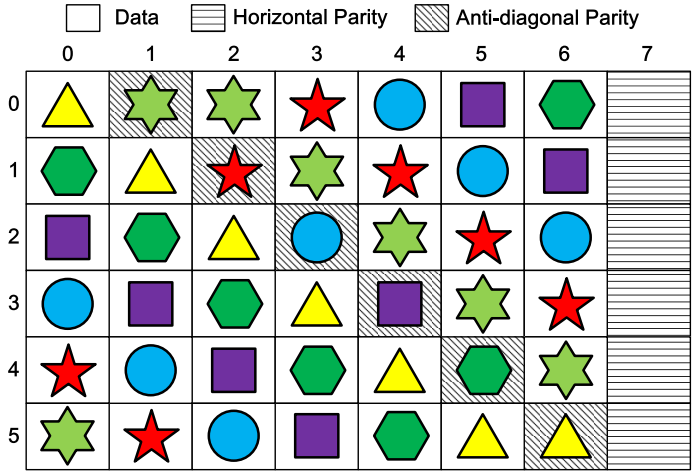
Anti-diagonal parity:

$$C_{i,i+1} = \sum_{j=0}^{p-1} C_{\langle p-2-i+j \rangle_p, j} \quad (j \neq i+1) \tag{2.3}$$

Figure 2.3 shows an example of H-Code for an 8-disk array ($p = 7$). It is a 6-row-8-column matrix. Column 7 is used for horizontal parity and the anti-diagonal elements ($C_{0,1}$, $C_{1,2}$, $C_{2,3}$, etc.) are used for anti-diagonal parity.



(a) Horizontal parity coding of H-Code: a horizontal element can be calculated by XOR operations among the corresponding data elements in the same row. For example, $C_{0,7} = C_{0,0} \oplus C_{0,2} \oplus C_{0,3} \oplus C_{0,4} \oplus C_{0,5} \oplus C_{0,6}$.



(b) Anti-diagonal parity coding of H-Code: an anti-diagonal element can be calculated by XOR operations among the corresponding data elements in all columns (except its column and column p). For example, $C_{1,2} = C_{4,0} \oplus C_{5,1} \oplus C_{0,3} \oplus C_{1,4} \oplus C_{2,5} \oplus C_{3,6}$.

Figure 2.3: H-Code ($p = 7$).

The horizontal parity encoding of H-Code is shown in Figure 2.3(a). We use different shapes to indicate different sets of horizontal elements and the corresponding data elements. Based on Equation 2.2, we calculate all horizontal elements. For example, the horizontal element $C_{0,7}$ can be calculated by $C_{0,0} \oplus C_{0,2} \oplus C_{0,3} \oplus C_{0,4} \oplus C_{0,5} \oplus C_{0,6}$. The element $C_{0,1}$ is not involved in this example because of $j = i + 1$.

The anti-diagonal parity encoding of H-Code is given in Figure 2.3(b). The anti-diagonal elements and their corresponding data elements are also distinguished by various shapes. According to Equation 2.3, the anti-diagonal elements can be calculated through modular arithmetic and XOR operations. For example, to calculate the anti-diagonal element $C_{1,2}$ ($i = 1$), first we should get the proper data elements ($C_{\langle p-2-i+j \rangle_p, j}$). If $j = 0$, by using Equation 2.3, $p-2-i+j = 4$ and then $\langle 4 \rangle_p = 4$, we get the first data element $C_{4,0}$. The following data elements which take part in XOR operations can be calculated similarly (the following data elements are $C_{5,1}$, $C_{0,3}$, $C_{1,4}$, $C_{2,5}$ and $C_{3,6}$). Second, the corresponding anti-diagonal element ($C_{1,2}$) is constructed by performing an XOR operation on these data elements, i.e., $C_{1,2} = C_{4,0} \oplus C_{5,1} \oplus C_{0,3} \oplus C_{1,4} \oplus C_{2,5} \oplus C_{3,6}$.

2.2.2 Construction Process

Based on the above data/parity layout and encoding scheme, the construction process of H-Code is straightforward.

- Label all data elements.
- Calculate both horizontal and anti-diagonal parity elements according to Equations 2.2 and 2.3.

2.2.3 Proof of Correctness

To prove that H-Code is correct, we consider one stripe. The reconstruction of multiple stripes is just a matter of scale and similar to the reconstruction of one stripe. In a stripe, we have the following lemma and theorem,

Lemma 2.1. *We can find a sequence of a two-integer tuple (T_k, T'_k) where*

$$T_k = \left\langle p - 1 + \frac{k+1+\frac{1+(-1)^k}{2}}{2}(f_2 - f_1) \right\rangle_p ,$$
$$T'_k = \frac{1+(-1)^k}{2} f_1 + \frac{1+(-1)^{k+1}}{2} f_2 \quad (k = 0, 1, \cdots, 2p-1)$$

with a prime number of p and $0 < f_2 - f_1 < p$, the endpoints are $(p-1, f_1)$ and $(p-1, f_2)$, and all two-integer tuples $(0, f_1)$, $(0, f_2)$, $\cdots$, $(p-1, f_1)$, $(p-1, f_2)$ occur exactly once in the sequence. Similar proof of this lemma can be found in many literatures in RAID-6 codes such as [8, 18, 86, 45].

Theorem 2.1. *A $(p-1)$-row-$(p+1)$-column stripe constructed according to the formal description of H-Code can be reconstructed under concurrent failures from any two columns.*

Proof. There are two cases of double failures, depending on whether column 0 fails or not.

Case I: column 0 doesn't fail.

There are further two subcases, depending on whether the horizontal parity column fails or not.

Case I-I: Double failures, one is from the horizontal parity column, and the other is from the anti-diagonal parity.

From the construction of H-Code, any two of the lost data elements and parity element are not in a same parity chain. Therefore, each of them can be recovered through the anti-diagonal parity chains. When all lost data elements are recovered, the horizontal parity elements can be reconstructed using the above Equation 2.2.

Case I-II: Double failures of any two columns other than the horizontal parity of the stripe.

Each column j, in the anti-diagonal parity part of the stripe, intersects all horizontal parity chains except the horizontal parity chain in the $j-1$th row. Therefore, each column misses a different horizontal parity chain.

First, we make an assumption that there is a pseudo row under the last row of H-Code matrix as shown in Figure 2.4(a). Each element of the additional row is all-zero-bit element and takes part into the generation of parity of its anti-diagonal, where it does not change the original value of anti-diagonal parity elements. This additional all-zero-bit element just participates in the generation of the parity element in its column. We assume that the two failed columns are f_1 and f_2, where $0 < f_1 < f_2 < p$.

From the construction of H-Code, each horizontal parity chain in the ith row intersects all columns in anti-diagonal part of the stripe except the $(i+1)$th column. Any two anti-diagonal parity elements cannot be placed in the same row. For any two concurrent failed columns f_1 and f_2, the two horizontal parity chains that are not intersected by both columns are in the (f_1-1)th row and the (f_2-1)th row. Since each of these horizontal parity chains only misses one data element, the missing element can be reconstructed along that horizontal parity chain with its horizontal parity. Since the horizontal parity column does not fail which means all horizontal parity elements are available, we can start reconstruction

process from the data element on each of the two missing columns using the horizontal parity.

For the failed columns f_1 and f_2, if a data element C_{i,f_2} on column f_2 can be reconstructed from the horizontal parity in horizontal parity chain in the ith row, we can reconstruct the missing data element $C_{\langle i+f_1-f_2\rangle_p,f_1}$ on the same anti-diagonal parity chain if we have its anti-diagonal parity element. Similarly, a data element C_{i,f_1} in column f_1 can be reconstructed from the horizontal parity in horizontal parity chain in the ith row, we can reconstruct the missing data element $C_{\langle i+f_2-f_1\rangle_p,f_2}$ on the same anti-diagonal parity chain if we have its anti-diagonal parity element parity element.

From the above discussion, the two missing anti-diagonal parity elements are just the two endpoints we mentioned in Lemma 3.1. If there are no missing parity elements, we start the reconstruction process from data element C_{f_1-1,f_2} on the f_2th column to the corresponding endpoint (element C_{p-1,f_1} on the f_1th column). In this reconstruction process, all data elements can be reconstructed and the reconstruction sequence is based on the sequence of the two-integer tuple in Lemma 3.1. Similarly, with no missing parity elements, we start the reconstruction process from data element C_{f_2-1,f_1} on the f_1th column to the corresponding endpoint (element C_{p-1,f_2} on the f_2th column).

However, the two missing anti-diagonal parity elements are not reconstructed in this case. After we start the reconstruction from the two data elements C_{f_1-1,f_2} and C_{f_2-1,f_1}, the missing anti-diagonal parity elements C_{p-1,f_1} and C_{p-1,f_2} cannot be recovered, because their corresponding horizontal parity and anti-diagonal parity are both missing. Actually, we do not need to reconstruct these two elements for they are not really in our code matrix.

In summary, all missing data elements are recoverable. After all data elements are recovered, we can reconstruct the two missing anti-diagonal parity elements.

Case II: column 0 fails.

This case is similar to *Case I*. The difference is that in *subcase II-II*, there is only one reconstruction sequence in reconstruction process. This process starts at $C_{f_2-1,0}$ and all lost data elements can be recovered. □

2.2.4 Reconstruction

We first consider how to recover a missing data element since any missing parity element can be recovered based on Equations 2.2 and 2.3. If we save the horizontal parity element and the related $p-2$ data elements, we can recover the missing data element (assume it's C_{i,f_1} in column f_1 and $0 \leq f_1 \leq p-1$) using the following equation,

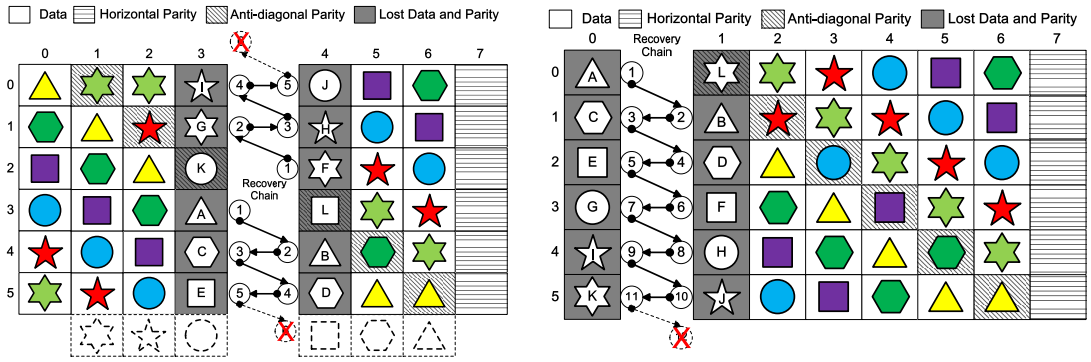
$$C_{i,f_1} = \sum_{j=0}^{p} C_{i,j} \quad (j \neq i+1 \quad and \quad j \neq f_1) \tag{2.4}$$

If there exists an anti-diagonal parity element and its $p-2$ data elements, to recover the data element (C_{i,f_1}), first we should find the corresponding anti-diagonal parity element. Assume it is in row r and this anti-diagonal parity element can be represented by $C_{r,r+1}$ based on Equation 2.3, we have,

$$r = \langle p-2-i+f_1 \rangle_p \tag{2.5}$$

And then according to Equation 2.3, the lost data element can be recovered,

$$\begin{aligned} &C_{i,f_1} = C_{r,r+1} \oplus \sum_{j=0}^{p-1} C_{\langle i-f_1+j \rangle_p, j} \\ &(j \neq f_1 \quad and \quad j \neq \langle p-1-i+f_1 \rangle_p) \end{aligned} \tag{2.6}$$



(a) Reconstruction by two recovery chains (there are double failures in columns 3 and 4): First we identify the two starting points of recovery chain: data elements A and F. Second we reconstruct data elements according to the corresponding recovery chains until they reach the endpoints (data elements E and J). The next anti-diagonal elements after E and J do not exist (C_{p-1,f_1} and C_{p-1,f_2} in the proof of Theorem 2.1, we use two "Xs" here), so the recovery chains end. The orders to recover data elements are: one is A→B→C→D→E, the other is F→G→H→I→J. Finally we reconstruct anti-diagonal parity elements K and L according to Equation 2.3.

(b) Reconstruction by one recovery chain (there are double failures in columns 0 and 1): First we identify the starting point of recovery chain: data element A. Second we reconstruct data elements according to the corresponding recovery chain until it reaches endpoint (data element K). The next anti-diagonal element after K does not exist (we use an "X" here), so the recovery chain ends. The order to recover data elements is: A→B→C→D→E→F→G→H→I→J→K. Finally we reconstruct the anti-diagonal parity element L according to Equation 2.3.

Figure 2.4: Reconstruction Process of H-Code.

Based on Equations 2.2 to 2.6, we can easily recover the elements with single disk failure. If two disks fail (for example, column f_1 and column f_2, $0 \leq f_1 < f_2 \leq p$), based on Theorem 2.1, we have our reconstruction algorithm of H-Code, shown in Algorithm 2.1.

As the proof of Theorem 2.1, there are two cases in our reconstruction algorithm of H-Code: failure in column 0 or not. Each has two subcases, *subcases I-I* and *II-I* focus on the scenario where at least one failure involves a horizontal parity column while in *subcases I-II* and *II-II*, failures don't involve the horizontal parity. The reconstruction examples of *subcases I-II* and *II-II* are shown in Figure 2.4(a) and Figure 2.4(b), which are situations with different numbers of recovery chains.

Algorithm 2.1: Reconstruction Algorithm of H-Code

Step 1: Identify the double failure columns: f_1 and f_2 ($f_1 < f_2$).
Step 2: Start reconstruction process and recover the lost data and parity elements.
switch $0 \leq f_1 < f_2 \leq p$ **do**
  case *I:* $f_1 \neq 0$ *(column* 0 *is saved)*
    case *I-I:* $f_2 = p$ *(horizontal parity column is lost)*
      Step 2-I-IA: Recover the lost data elements in column f_1.
      repeat
        Compute the lost data elements (C_{i,f_1}, $i \neq f_1 - 1$) based on Equations 2.5 and 2.6.
      until *all lost data elements are recovered.*
      Step 2-I-IB: Recover the lost anti-diagonal parity element (C_{f_1-1,f_1}) based on Equation 2.3.
      Step 2-I-IC: Recover the lost horizontal parity elements in column f_2.
      repeat
        Compute the lost horizontal parity elements (C_{i,f_2}) based on Equation 2.2.
      until *all lost horizontal parity elements are recovered.*

    case *I-II:* $f_2 \neq p$ *(horizontal parity column is saved)*
      Step 2-I-IIA: Compute two starting points (C_{f_2-1,f_1} and C_{f_1-1,f_2}) of the recovery chains based on Equation 2.4.
      Step 2-I-IIB: Recover the lost data elements in the two recovery chains.
      Two cases start synchronously:
      case *starting point is* C_{f_2-1,f_1} **repeat**
        (1) Compute the next lost data element (in column f_2) in the recovery chain based on Equations 2.5 and 2.6;
        (2) Then compute the next lost data element (in column f_1) in the recovery chain based on Equation 2.4.
      until *at the endpoint of the recovery chain.*
      case *starting point is* C_{f_1-1,f_2} **repeat**
        (1) Compute the next lost data element (in column f_1) in the recovery chain based on Equations 2.5 and 2.6;
        (2) Then compute the next lost data element (in column f_2) in the recovery chain based on Equation 2.4.
      until *at the endpoint of the recovery chain.*
      Step 2-I-IIC: Recover the lost anti-diagonal parity element in column f_1 and f_2.
      repeat
        Compute the lost anti-diagonal parity elements (C_{f_1-1,f_1} and C_{f_2-1,f_2}) based on Equation 2.3.
      until *all lost anti-diagonal parity elements are recovered.*

  case *II:* $f_1 = 0$ *(column* 0 *is lost)*
    case *II-I:* $f_2 = p$ *(horizontal parity column is lost)*
      Step 2-II-IA: Recover the lost data elements in column 0.
      repeat
        Compute the lost data elements ($C_{i,0}$) based on Equations 2.5 and 2.6.
      until *all lost data elements are recovered.*
      Step 2-II-IB: Recover the lost horizontal parity elements in column f_2.
      repeat
        Compute the lost horizontal parity elements (C_{i,f_2}) based on Equation 2.2.
      until *all lost horizontal parity elements are recovered.*

    case *II-II:* $f_2 \neq p$ *(horizontal parity column is saved)*
      Step 2-II-IIA: Compute the starting point of the recovery chain ($C_{f_2-1,0}$) based on Equation 4.
      Step 2-II-IIB: Recover the lost data elements in the recovery chain.
      repeat
        (1) Compute the next lost data element in column f_2 based on Equations 2.5 and 2.6;
        (2) Then compute the next lost data element in column 0 based on Equation 2.4.
      until *at the endpoint of the recovery chain.*
      Step 2-II-IIC: Recover the lost diagonal parity element in column f_2.
      Compute the lost anti-diagonal parity element (C_{f_2-1,f_2}) based on Equation 2.3.

2.3 Property Analysis

In this section, we first prove that H-Code shares some optimal properties as other vertical codes, including: optimal storage efficiency, optimal encoding/decoding computational complexity, optimal single write cost. Then, we prove that H-Code has optimal complexity of partial stripe write to two data elements in the same row. Furthermore, H-Code has optimal complexity of partial stripe write to two data elements in a stripe. Finally, we evaluate partial stripe write cost of different codes in two aspects: in the same row and across two rows.

2.3.1 Optimal Property of H-Code

Vertical codes have the optimal storage efficiency, optimal encoding/decoding computational complexity, optimal single write complexity [86, 15, 45]. We will prove that H-Code shares the optimal property as other vertical codes.

Optimal Storage Efficiency

From the proof of H-Code's correctness, H-Code is a MDS code. Since all MDS codes have optimal storage efficiency [86, 15, 45], H-Code is storage efficient. Because H-Code doesn't use a dedicated anti-diagonal strip, it will not suffer from the unbalanced I/O distribution as evidenced by our results shown in Figure 2.10, which will be discussed in more detail later. By shifting the stripes' horizontal parity strips among all the disks just as RAID-5, H-Code does not suffer from the intensive I/O operations on dedicated parity disk caused by random writes among stripes, either.

Optimal Encoding/Decoding Computational Complexity

From the construction of H-Code, to generate all the $2*(p-1)$ parity elements in a $(p-1)$-row-$(p+1)$-column constructed H-Code, each of the remaining $(p-1)*(p-1)$ data elements needs to take part into two XOR operations. Thus, the encoding computational complexity of H-Code is $[2*(p-1)*(p-1)-2*(p-1)]/[(p-1)*(p-1)]$ XOR operations per data element on average. To reconstruct $2*(p-1)$ failed elements in the case of double disk failures, it need use $2*(p-1)$ parity chains. Every parity chain in H-Code has the same length of $(p-1)$. Thus, the decoding computational complexity of H-Code is $(p-3)$ XOR operations per lost element on average. P-Code [45] has already proved that an i-row-j-column constructed code with x data elements, has an optimal encoding computational complexity of $(3x-i*j)/x$ XOR operations per data element on average, and decoding computational complexity of $(3x-i*j)/(i*j-x)$ XOR operations per lost element on average. Therefore, H-Code's encoding/decoding computational complexity is optimal.

Optimal Single Write Property

From the construction of H-Code, each of the data elements takes part into the generation of two and only two parity elements. Therefore, a single write on one data element in H-Code only causes one additional write on each of the two parity elements, which has been proved to be optimal in previous research [86, 15, 45].

2.3.2 Partial Stripe Writes to Two Continuous Data Elements in H-Code

Now, we prove that the cost of any partial stripe write to two continuous data elements of H-Code is optimal among all vertical lowest density MDS codes.

Theorem 2.2. *Any two data elements in a stripe of a lowest density vertical MDS code are in at least three parity chains.*

Proof. From previous literatures [86, 15, 45], it has been proved that, in a stripe of a lowest density MDS code, any one data element takes part into the generation of two and only two parity elements. Assume there are two data elements in two or less parity chains. Consider the following case: both data elements are lost when double disk failures occur. Now, these two lost data elements are unrecoverable, because all parity chains in the code include either messages of both of these two data elements or none of them. Thus, the assumption is invalid. Therefore, in a stripe of a lowest density MDS code, any two data elements should be in at least three parity chains. □

From the construction, any two continuous data elements in the same row of H-Code share a same horizontal parity and must not share the same anti-diagonal parity. In other words, any two continuous data elements in the same row are in three different parity chains including a horizontal parity chain and two different anti-diagonal parity chains. From Equation 2.1, any partial stripe writes to two continuous data elements in a row of H-Code causes $2 * (2 + 3) = 10$ I/O operations.

Furthermore, as shown in Figure 2.5, from the construction of H-Code, any two continuous data elements across two different rows share the same anti-diagonal parity

and must not share a same horizontal parity. From Equation 2.1, in this condition H-Code also causes 10 I/O operations per partial stripe write.

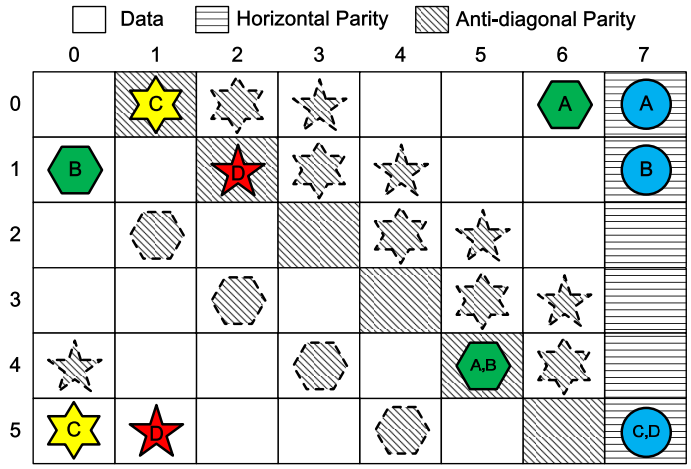


Figure 2.5: Partial stripe writes to two continuous data elements in H-Code for an 8-disk array.

In this figure, a partial stripe write to data elements A and B in two different rows, and the other partial stripe write to data elements C and D in the same row. In these two cases, there are only 3 parity elements modified for each partial stripe write, which shows that our H-Code reduces partial stripe write cost and improves the performance of storage system.

In summary, any partial stripe write to two continuous data elements of H-Code are in three different parity chains. This is the lowest bound we proved in Theorem 2.2. From Equation 2.1 and Theorem 2.2, the cost of any partial stripe write to two continuous data elements locally of H-Code is 10 I/O operations, which is optimal.

2.3.3 Different Cases of Partial Stripe Writes to w Continuous Data Elements

There are two cases for a partial stripe write to w continuous data elements: one is the w written continuous data elements in the same row, the other is these data elements across rows.

The case of partial stripe writes to w continuous data elements ($2 \leq w \leq n-3$) in the same row is very simple, we only need to calculate the number of parity chains based on Equation 2.1. For example, when a partial stripe write to w continuous data elements in H-code, these data elements to be written share one horizontal parity chain, but are in w different anti-diagonal parity chains. According to Equation 2.1, the total I/O operations are $(4w+2)$.

However, for a partial stripe write to w continuous data elements ($2 \leq w \leq n-3$), there are many scenarios where partial stripe writes are crossing different rows (will be discussed in Section 2.4), which are not as simple as partial stripe writes to two continuous data elements. For example, as shown in Figure 2.1, if $w = 3$, the cost of a partial stripe write to three elements $C_{0,4}C_{0,5}C_{1,0}$ in RDP is different from a partial stripe write to $C_{1,4}C_{1,5}C_{2,0}$.

2.4 Performance Evaluation

In this section, we give our evaluations to demonstrate the effectiveness of our H-Code for partial stripe writes to w continuous data elements ($2 \leq w \leq n-3$).

2.4.1 Evaluation Methodology

We compare H-Code with following popular codes in typical scenarios when $p = 5$ and $p = 7$ (P-Code has two variations, which are denoted by P-Code-1 and P-Code-2):

(1) **Codes for $p-1$ disks**: P-Code-1 [45] and Cyclic code [15];

(2) **Codes for p disks**: X-Code [86] and P-Code-2 [45];

(3) **Codes for $p+1$ disks**: H-Code and RDP code [18];

(4) **Codes for $p+2$ disks**: EVENODD code [8].

To reflect the status of partial stripe writes among different codes, we envision an ideal sequence to partial stripe writes as follows,

For each data element, it is treated as the beginning written element at least once in a partial stripe write to w continuous data elements ($2 \leq w \leq n-3$, including partial stripe writes in the same row and across two rows). If there is no data element at the end of a stripe, the data element at the beginning of the stripe will be written.

In our evaluation, a partial stripe write across different stripes is a little different from the one in our ideal sequence, which need more I/O operations but has little effect on the evaluation results. When $p = 5$ and $p = 7$ in different codes, the number of data elements in a stripe is less than $7 * 7 = 49$.

Based on the description of ideal sequence, for H-Code shown in Figure 2.3 (which has $(p-1)^2$ total data elements in a stripe), the ideal sequence to w partial stripe writes is: $C_{0,0}C_{0,2}\cdots$, $C_{0,2}C_{0,3}\cdots$, $C_{0,3}C_{0,4}\cdots$, and so on, $\cdots\cdots$, the last partial stripe write in this sequence is $C_{p-2,p-2}C_{0,0}\cdots$.

We use two types of access patterns based on this ideal sequence,

(1) Uniform access. Each partial stripe write occurs only once, so each data element is written w times.

(2) Random access. Since the number of total data elements in a stripe is less than 49 when $p = 5$ and $p = 7$, we use 50 random numbers (ranging from 1 to 1000) generated by a random integer generator [66] as the frequencies of partial stripe writes in the sequence one after another. These 50 random numbers are shown in Table 3.2. For example, the first

Table 2.1: 50 Random Integer Numbers

221	811	706	753	34	862	353	428	99	502
969	800	32	346	889	335	361	209	609	11
18	76	136	303	175	71	427	143	870	855
706	297	50	824	324	212	404	199	11	56
822	301	430	558	954	100	884	410	604	253

number in the table,"221" is used as the frequency of the first stripe writes $C_{0,0}C_{0,2}\cdots$ (for H-Code).

$F(C_{i,j})$ and $P(C_{i,j})$ are used to denote the access frequency and the access probability of a partial stripe write to w continuous data elements starting from $C_{i,j}$, respectively. If N_s is the total access frequency of all stripe writes in the ideal sequence, then,

$$P(C_{i,j}) = \frac{F(C_{i,j})}{N_s}, \quad \sum P(C_{i,j}) = 1 \tag{2.7}$$

For example, in uniform access, the access frequency and access probability of any stripe write are,

$$F(C_{i,j}) = 1, \quad P(C_{i,j}) = \frac{1}{N_s} \tag{2.8}$$

We evaluate H-Code and other codes in terms of the following metrics. The first metric is denoted by $S_w(C_{i,j})$, which is the number of I/O operations a partial stripe write to w continuous data elements starting from $C_{i,j}$. We define "**average number of I/O operations of a partial stripe write to w continuous data elements** ($S_{avg.}(w)$)" to evaluate different codes. The smaller value of $S_{avg.}(w)$ is, the lower cost of partial stripe writes and the higher performance of storage system is. $S_{avg.}(w)$ can be calculated by,

$$S_{avg.}(w) = \sum S_w(C_{i,j}) \cdot P(C_{i,j}) \tag{2.9}$$

For uniform access in H-Code, $N_s = (p-1)^2$. According to Equation 2.8, the average number of I/O operations of the ideal sequence of partial stripe writes is calculated using the following equation,

$$S_{avg.}(w) = \frac{\sum_{i=0}^{p-2} \left[\sum_{j=0}^{p-1} S_w(C_{i,j}) \right]}{(p-1)^2} \quad (j \neq i+1) \tag{2.10}$$

As described in Section 2.3.2 and Figure 2.5, it takes 10 I/O operations for any partial stripe write to two continuous data elements. According to Equation 2.12, we have the $S_{avg.}(w)$ value of H-Code,

$$S_{avg.}(w=2) = \frac{10(p-1)^2}{(p-1)^2} = 10$$

According to Equation 2.1, for a random partial stripe write, the numbers of read and write I/O are the same, so we use average number of I/O operations to evaluate different codes.

Our next metric is the "**maximum number of I/O operations of a partial stripe write to w continuous data elements** ($S_{max.}(w)$)". It is the maximum number of I/O operations of all partial stripe writes in the sequence and calculated by,

$$S_{max.}(w) = \max\left[S_w(C_{i,j})\right] \tag{2.11}$$

To show the I/O distribution among different disks, we use another metric, $S_w^j(C_{i,j})$, to denote the number of I/O operations in column j of a partial stripe write to w continuous

data elements starting from $C_{i,j}$. We define "**average number of I/O operations in column j of a partial stripe write to w continuous data elements** ($S^j_{avg.}(w)$)" as follows,

$$S^j_{avg.}(w) = \sum S^j_w(C_{i,j}) \cdot P(C_{i,j}) \tag{2.12}$$

For H-Code, we have the following equation,

$$S^j_{avg.}(w) = \frac{\sum_{i=0}^{p-2} \left[\sum_{j=0}^{p-1} S^j_w(C_{i,j}) \right]}{(p-1)^2} \quad (j \neq i+1) \tag{2.13}$$

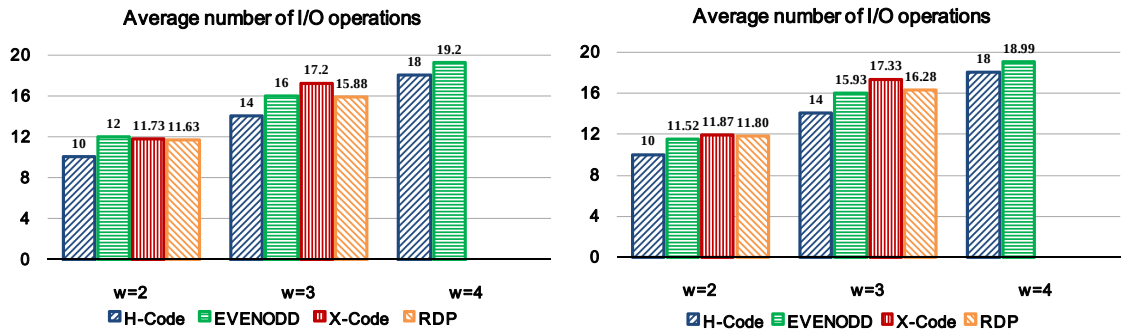
2.4.2 Numerical Results

In this subsection, we give the numerical results of H-Code compared to other typical codes using above metrics. In the following figures and tables, due to the constraint of $2 \leq w \leq n-3$, the average/maximum number of I/O operations in some codes are not available.

Average I/O Operations

First we calculate the average I/O operation counts ($S_{avg.}(w)$ values) for different codes with various w and p shown in Figure 2.6 and Figure 2.7. The results show that H-Code can reduce the cost of partial stripe writes, and thus improve the performance of storage system.

We also summarize the costs in terms of I/O operations of our H-Code compared to other codes, which are shown in Table 2.2 and 2.3. It is obvious that H-Code has the lowest I/O cost of partial stripe writes. For uniform access, there is a decrease of I/O operations up to 14.68% and 20.45% compared to RDP and EVENODD codes, respectively. For random



(a) uniform access

(b) random access

Figure 2.6: Average number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 5$.
In this figure, 5 disks are used for X-Code, 6 disks are used for H-Code and RDP code, and 7 disks are used for EVENODD code.

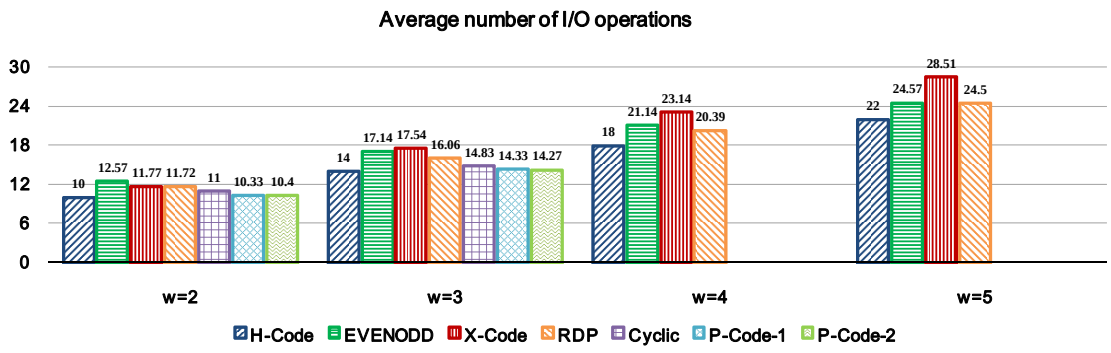
Table 2.2: Improvement of H-Code over Other Codes in terms of Average Partial Stripe Write Cost (Uniform Access)

w & p	EVENODD code	X-Code	RDP code	Cyclic code	P-Code-1	P-Code-2
$w = 2, p = 5$	16.67%	14.75%	14.02%	–	–	–
$w = 3, p = 5$	12.50%	18.60%	11.84%	–	–	–
$w = 2, p = 7$	20.45%	15.04%	14.68%	9.09%	3.19%	3.85%
$w = 3, p = 7$	18.32%	20.18%	12.83%	5.60%	2.30%	1.89%
$w = 4, p = 7$	14.85%	22.21%	11.72%	–	–	–
$w = 5, p = 7$	10.46%	22.83%	10.20%	–	–	–

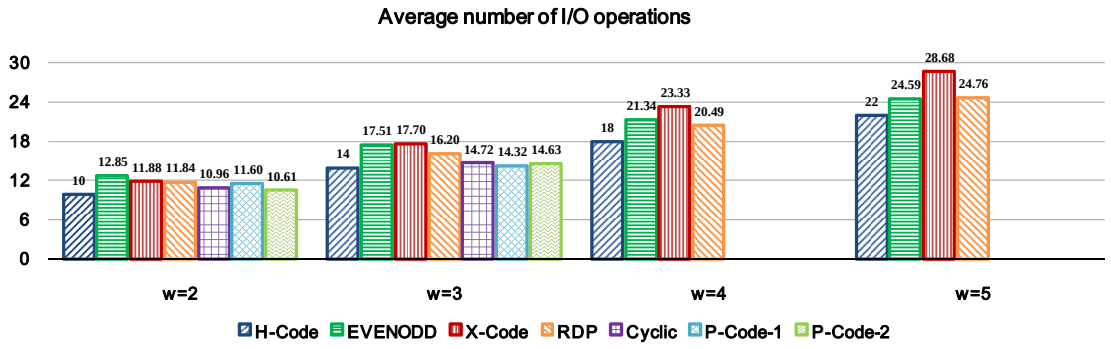
access, compared to RDP and EVENODD codes, H-Code reduces the cost by up to 15.54% and 22.17%, respectively.

Maximum I/O Operations

Next we evaluate the maximum I/O operations shown in Figure 2.8 and 2.9. It clearly shows that H-Code has the lowest maximum number of I/O operations compared to other coding methods in all cases.



(a) uniform access



(b) random access

Figure 2.7: Average number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 7$.
In this figure, 6 disks are used for P-Code-1 and Cyclic code, 7 disks are used for X-Code and P-Code-2, 8 disks are used for H-Code and RDP code, and 9 disks are used for EVENODD code.

The above evaluations demonstrate that H-Code outperforms other codes in terms of average and maximum I/O operations. The reasons that H-Code has the lowest partial stripe write cost are: First, H-Code has a special anti-diagonal parity (the last data element in a row and the first data element in the next row share the same anti-diagonal parity) , which can decease the partial stripe write cost when continuous data elements are crossing rows like vertical codes. Second, we keep the horizontal parity similar to EVENODD and RDP codes, which is efficient for partial stripe writes to continuous data elements in

Table 2.3: Improvement of H-Code over Other Codes in terms of Average Partial Stripe Write Cost (Random Access)

w & p	EVENODD code	X-Code	RDP code	Cyclic code	P-Code-1	P-Code-2
$w = 2, p = 5$	13.19%	15.75%	15.25%	–	–	–
$w = 3, p = 5$	12.12%	19.22%	14.00%	–	–	–
$w = 2, p = 7$	22.17%	15.82%	15.54%	8.76%	13.79%	5.75%
$w = 3, p = 7$	20.04%	20.90%	13.58%	4.89%	2.23%	4.31%
$w = 4, p = 7$	15.65%	22.84%	12.15%	–	–	–
$w = 5, p = 7$	10.53%	23.29%	11.15%	–	–	–

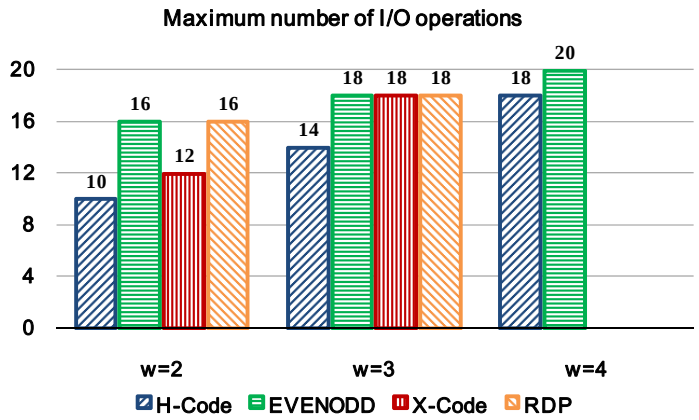


Figure 2.8: Maximum number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 5$.
In this figure, 5 disks are used for X-Code, 6 disks are used for H-Code and RDP code, and 7 disks are used for EVENODD code.

the same row. Therefore, our H-Code takes advantages of both horizontal codes (such as EVENODD and RDP) and vertical codes (such as X-Code, Cyclic and P-Code).

Partial Stripe Write in the Same Row

Third, we evaluate the cost for a partial stripe write to w continuous data elements in the same row of H-Code and some other typical codes shown in Table 2.4. Compared to EVENODD and X-Code, H-Code reduces I/O cost by up to 19.66% and 16.67%, respectively.

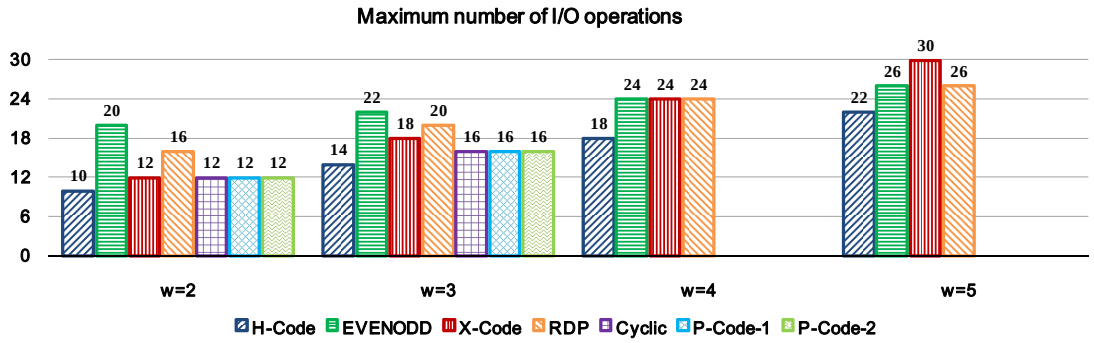


Figure 2.9: Maximum number of I/O operations of a partial stripe write to w continuous data elements of different codes with different value of w when $p = 7$.
In this figure, 6 disks are used for P-Code-1 and Cyclic code, 7 disks are used for X-Code and P-Code-2, 8 disks are used for H-Code and RDP code, and 9 disks are used for EVENODD code.

Table 2.4: Cost Of A Partial Stripe Write to w Continuous Data Elements in the Same Row ($p = 7$, Uniform Access)

	H-Code	EVENODD	X-Code
$w = 2$	10	12.44	12
$w = 3$	14	16.8	16
$w = 4$	18	20.5	20

From Table 2.4, we find that, for a partial stripe write within a row, H-code offers better partial stripe write performance compared to other typical codes. Due to the horizontal parity, H-code performs better than X-Code (e.g., 10 vs. 12 I/O operations when $w = 2$) in partial stripe write performance in the same row. H-Code has much lower cost than EVENODD because of different anti-diagonal construction schemes.

I/O Workload Balance

As we mentioned before, H-Code doesn't suffer from unbalanced I/O distribution which is an issue in RDP code. To verify this, we calculate the $S^j_{avg.}(w)$ values for H-Code and RDP code as shown in Figure 2.10. It shows that for RDP code, the workload is not balance (in disk 6 and disk 7 it is very high, especially in disk 7). However, our H-Code balances

the workload very well among all disks because of the dispersed anti-diagonal parity in different columns.

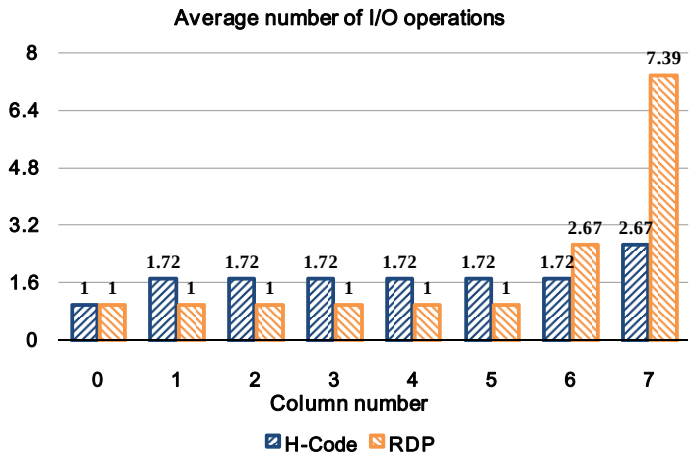


Figure 2.10: Average number of I/O operations in column j of a partial stripe write to three continuous data elements of different codes in an 8-disk array ($p = 7, w = 3$).

2.5 Summary

In this chapter, we propose a Hybrid Code (H-Code), to optimize partial stripe writes for RAID-6 in addition to take advantages of both horizontal and vertical MDS codes. H-Code is a solution for an array of $(p+1)$ disks, where p is a prime number. The parities in H-Code include horizontal row parity and anti-diagonal parity, where anti-diagonal parity elements are distributed among disks in the array. Its horizontal parity ensures a partial stripe write to continuous data elements in a row share the same row parity chain to achieve optimal partial stripe write performance. Our theoretical analysis shows that H-Code is optimal in terms of storage efficiency, encoding/decoding computational complexity and single write complexity. Furthermore, we find H-Code offers optimal partial stripe write complexity to two continuous data elements and optimal partial stripe write performance among all

MDS codes to the best of our knowledge. Our H-code reduces partial stripe write cost by up to 15.54% and 22.17% compared to RDP and EVENODD codes. In addition, H-Code achieves better I/O workload balance compared to RDP code.

Chapter 3

HDP Code

3.1 Introduction

As mentioned in previous chapters, a typical RAID-6 storage system based on horizontal codes is composed of $k + 2$ disk drives. The first k disk drives are used to store original data, and the last two are used as parity disk drives. Horizontal codes have a common disadvantage that k elements must be read to recover any one other element. Vertical codes have been proposed that disperse the parity across all disk drives, including X-Code [86], Cyclic code [15], and P-Code [45]. All MDS codes have a common disadvantage that k elements must be read to recover any one other element. This limitation reduces the reconstruction performance during single disk or double disk failures.

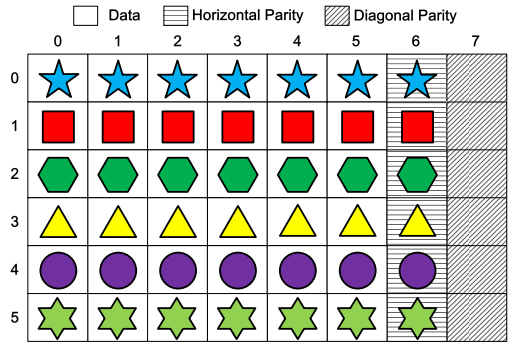
However, most MDS codes based RAID-6 systems suffer from unbalanced I/O, especially for write-intensive applications. Typical horizontal codes have dedicated parities, which need to be updated for any write operations, and thus cause higher workload on parity disks. For example, Figure 3.1 shows the load balancing problem in RDP [18] and the horizontal parity layout shown in Figure 3.1(a). Assuming $C_{i,j}$ delegates the element in

ith row and jth column, in a period there are six reads and six writes to a stripe as shown in Figure 3.1(c). For a single read on a data element, there is just one I/O operation. However, for a single write on a data element, there are at least six I/O operations, one read and one write on data elements, two read and two write on the corresponding parity elements. Then we can calculate the corresponding I/O distribution and find that it is an extremely unbalanced I/O as shown in Figure 3.1(d). The number of I/O operations in column 6 and 7 are five and nine times higher than column 0, respectively. It may lead to a sharp decrease of reliability and performance of a storage system.

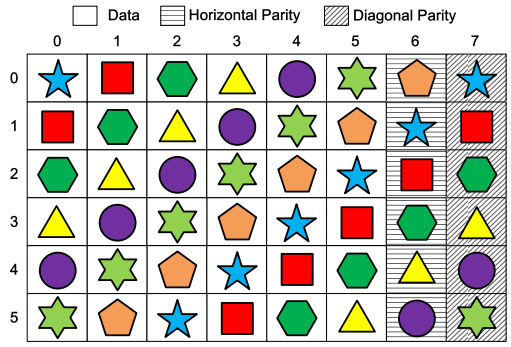
Unbalanced I/O also occurs on some vertical codes like P-Code [45] consisting of a prime number of disks due to unevenly distributed parities in a stripe. For example, Figure 3.2 shows the load balancing problem in P-Code [45]. From the layout of P-Code as shown in Figure 3.2(a), column 6 goes without any parity element compared to the other columns, which leads to unbalanced I/O as shown in Figure 3.2(b) though uniform access happens. We can see that column 6 has very low workload while column 0's workload is very high (six times of column 6). This can decrease the performance and reliability of the storage system.

Even if many dynamic load balancing approaches [26, 71, 44, 5, 36, 6] are given for disk arrays, it is still difficult to adjust the high workload in parity disks and handle the override on data disks. Although some vertical codes such as X-Code [86] can balance the I/O but they have high cost to recover single disk failure.

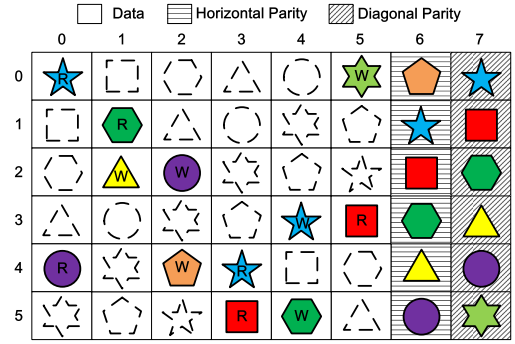
The unbalanced I/O hurts the overall storage system performance and the original single/double disk recovery method has some limitation to improve the reliability. To address this issue, we propose a new parity called **H**orizontal-**D**iagonal **P**arity (HDP) , which takes advantage of both horizontal parity and diagonal/anti-diagonal parity



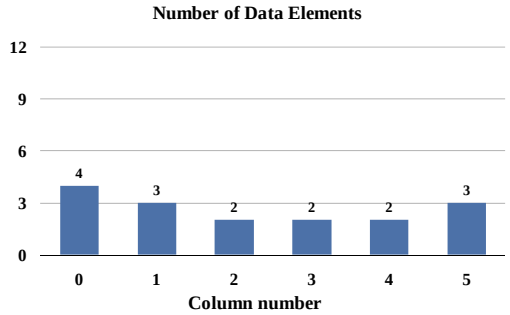
(a) Horizontal parity layout of RDP code with *prime+1* disks ($p = 7$).



(b) Diagonal parity layout of RDP code with *prime+1* disks ($p = 7$).



(c) Six reads and six writes to various data elements in RDP (Data elements $C_{0,0}$, $C_{1,1}$, $C_{3,5}$, $C_{4,0}$, $C_{4,3}$ and $C_{5,3}$ are read and other data elements $C_{0,5}$, $C_{2,1}$, $C_{2,2}$, $C_{3,4}$, $C_{4,2}$ and $C_{5,4}$ are written, which is a 50% read 50% write mode).

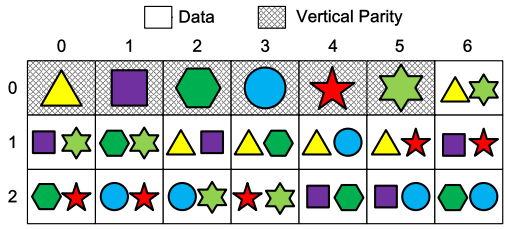


(d) The corresponding I/O distribution among different columns (e.g., the I/O operations in column 1 is 2 reads and 1 write, 3 operations in total; the I/O operations in disk 6 is 6 reads and 6 writes, 12 operations in total; the I/O operations in disk 7 is 10 reads and 10 writes, 20 operations in total).

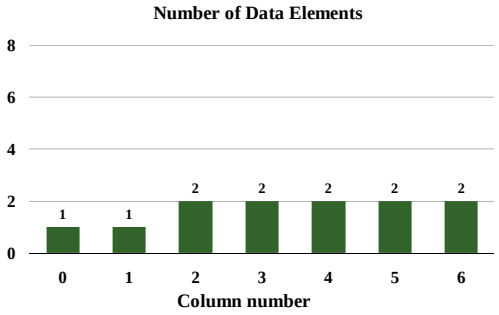
Figure 3.1: Load balancing problem in RDP code for an 8-disk array.

In this figure, the I/O operations in columns 0, 1, 2, 3, 4 and 5 are very low, while in columns 6 and 7 are very high. These high workload in parity disks may lead to a sharp decrease of reliability and performance of a storage system.

to achieve well balanced I/O. The corresponding code using HDP parities is called **H**orizontal-**D**iagonal **P**arity Code (HDP Code) and distributes all parities evenly to achieve balanced I/O. Depending on the number of disks in a disk array, HDP Code is a solution for $p - 1$ disks, where p is a prime number.



(a) Vertical parity layout of P-Code with *prime* disks ($p = 7$).



(b) I/O distribution among different columns when six writes to different columns occur (Continuous data elements $C_{0,6}$, $C_{1,0}$, $C_{1,1}$, $C_{1,2}$, $C_{1,3}$, $C_{1,4}$ and $C_{1,5}$ are written, each column has just one write to data element and it is a uniform write mode to various disks).

Figure 3.2: Load balancing problem in P-Code for a 7-disk array.

In this figure, the I/O operations in columns 0 and 5 are very high which also may lead to a sharp decrease of reliability and performance of a storage system.

The rest of this chapter continues as follows: Section 3.2 discusses the motivation. HDP Code is described in detail in Section 3.3. Load balancing and reliability analysis are given in Section 3.4 and 3.5. Finally we conclude the chapter in Section 3.6.

3.2 Summary on Load Balancing of Various Parities and Our Motivation

To find out the root of unbalanced I/O in various MDS coding approaches, we analysis the features of different parities, which can be classified into three categories: horizontal parity (row parity), diagonal/anti-diagonal parity (special vertical parity) and vertical parity.

3.2.1 Horizontal Parity (HP)

Horizontal Parity is the most important feature for all horizontal codes, such as EVENODD [8] and RDP [18], etc. The horizontal parity layout is shown in Figure 3.1(a) and can be calculated by,

$$C_{i,n-2} = \sum_{j=0}^{n-3} C_{i,j} \tag{3.1}$$

From Equation 3.1, we can see that typical HP can be calculated by some simple XOR computations. For a partial stripe write to continuous data elements, it could have lower cost than other parities due to these elements can share the same HP (e.g., partial write cost to continuous data elements $C_{2,1}$ and $C_{2,2}$ shown in Figure 3.1(c)). Through the layout of HP, horizontal codes can be optimized to reduce the recovery time when data disk fails [84].

However, there is an obvious disadvantage for HP, which is workload unbalance as introduced in Section 3.1.

3.2.2 Diagonal/Anti-diagonal Parity (DP/ADP)

Diagonal or anti-diagonal parity typically appears in horizontal codes or vertical codes, such as in RDP [18] and X-Code [86], which are shown in Figure 3.1(b) and 1.5 and keep balance well. The anti-diagonal parity of X-Code can be calculated by,

$$C_{n-1,i} = \sum_{k=0}^{n-3} C_{k,\langle i-k-2\rangle_n} \tag{3.2}$$

From Equation 3.2, some modular computation to calculate the corresponding data elements (e.g., $\langle i-k-2\rangle_n$) are added compared to Equation 3.1.

For DP/ADP, it has a little effect to reduce the single disk failure as discussed in Section 1.3.3.

3.2.3 Vertical Parity (VP)

Vertical parity is normally in vertical codes, such as in B-Code [85] and P-Code [45]. Figure 3.2(a) shows the layout of vertical parity. The construction of vertical parity is a little more complex: first some data elements are selected as a set and then do the modular calculation. So the computation cost for a vertical parity is higher than other parities. Except for some variations like P-Code shown in Figure 3.2(a), most vertical codes keep balance well. Due to the complex layout, vertical codes like P-Code are too hard to reduce the I/O cost of any single disk.

3.2.4 Motivation

Table 3.1 summarizes different parities, which all suffer from unbalanced I/O and the efficiency to reduce the I/O cost of single disk failure. To address these issues, we propose a new parity named HDP to take advantage of both vertical and horizontal parities to offer low computation cost, high reliability and balanced I/O. The layout of HDP is shown in Figure 3.3(a). In next section we will discuss how to use HDP parity to build HDP code to achieve I/O balance and high reliability.

Table 3.1: Summary of Different Parities

Parities	Computation cost	Workload balance	Reduce I/O cost of single disk failure
HP	very low	unbalance	a lot for data disks, little for parity disks
DP/ADP	medium	balance	some for all disks
VP	high	mostly balance (unbalance in P-Code)	none

3.3 HDP Code

To overcome the shortcomings of existing MDS codes, in this section we propose the HDP Code, which takes the advantage of both both horizontal and diagonal/anti-diagonal parities in MDS codes. HDP is a solution for $p-1$ disks, where p is a prime number.

3.3.1 Data/Parity Layout and Encoding of HDP Code

HDP Code is composed of a $p-1$-row-$p-1$-column square matrix with a total number of $(p-1)^2$ elements. There are three types of elements in the square matrix: ***data elements***, ***horizontal-diagonal parity elements***, and ***anti-diagonal parity elements***. $C_{i,j}$ $(0 \leq i \leq p-2,\ 0 \leq j \leq p-2)$ denotes the element at the ith row and the jth column. Two diagonals of this square matrix are used as horizontal-diagonal parity and anti-diagonal parity, respectively.

Horizontal-diagonal parity and anti-diagonal parity elements of HDP Code are constructed according to the following encoding equations:

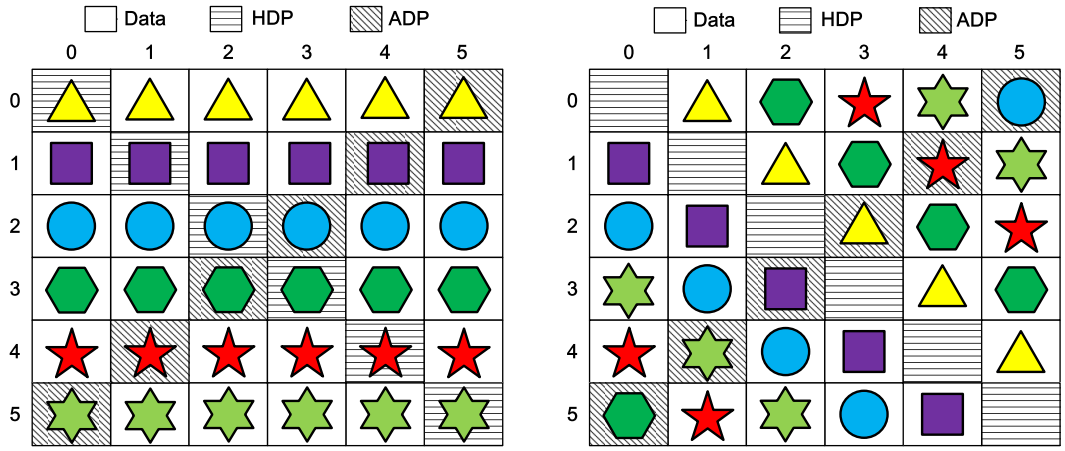
Horizontal parity:

$$C_{i,i} = \sum_{j=0}^{p-2} C_{i,j} \quad (j \neq i) \tag{3.3}$$

Anti-diagonal parity:

$$C_{i,p-2-i} = \sum_{j=0}^{p-2} C_{\langle 2i+j+2 \rangle_p, j}$$
$$(j \neq p-2-i \quad and \quad j \neq \langle p-3-2i \rangle_p) \tag{3.4}$$

Figure 3.3 shows an example of HDP Code for an 6-disk array ($p = 7$). It is a 6-row-6-column square matrix. Two diagonals of this matrix are used for the horizontal-diagonal parity elements ($C_{0,0}$, $C_{1,1}$, $C_{2,2}$, etc.) and the anti-diagonal parity elements ($C_{0,5}$, $C_{1,4}$, $C_{2,3}$, etc.).



(a) Horizontal-diagonal parity coding of HDP Code: a horizontal-diagonal parity element can be calculated by XOR operations among the corresponding data elements in the same row. For example, $C_{0,0} = C_{0,1} \oplus C_{0,2} \oplus C_{0,3} \oplus C_{0,4} \oplus C_{0,5}$.

(b) Anti-diagonal parity coding of HDP Code: an anti-diagonal parity element can be calculated by XOR operations among the corresponding data elements in anti-diagonal. For example, $C_{1,4} = C_{4,0} \oplus C_{5,1} \oplus C_{0,3} \oplus C_{2,5}$.

Figure 3.3: HDP Code ($p = 7$).

The encoding of the horizontal-diagonal parity of HDP Code is shown in Figure 3.3(a). We use different icon shapes to denote different sets of horizontal-diagonal elements and the corresponding data elements. Based on Equation 3.3, all horizontal

elements are calculated. For example, the horizontal element $C_{0,0}$ can be calculated by $C_{0,1} \oplus C_{0,2} \oplus C_{0,3} \oplus C_{0,4} \oplus C_{0,5}$.

The encoding of anti-diagonal parity of HDP Code is given in Figure 3.3(b). According to Equation 3.3, the anti-diagonal elements can be calculated through modular arithmetic and XOR operations. For example, to calculate the anti-diagonal element $C_{1,4}$ ($i = 1$), first we should fetch the proper data elements ($C_{\langle 2i+j+2 \rangle_p, j}$). If $j = 0$, $2i + j + 2 = 4$ and $\langle 4 \rangle_p = 4$, we get the first data element $C_{4,0}$. The following data elements, which take part in XOR operations, can be calculated in the same way (the following data elements are $C_{5,1}$, $C_{0,3}$, $C_{2,5}$). Second, the corresponding anti-diagonal element ($C_{1,4}$) is constructed by an XOR operation on these data elements, i.e., $C_{1,4} = C_{4,0} \oplus C_{5,1} \oplus C_{0,3} \oplus C_{2,5}$.

3.3.2 Construction Process

According to the above data/parity layout and encoding scheme, the construction process of HDP Code is straightforward:

- Label all data elements.
- Calculate both horizontal and anti-diagonal parity elements according to Equations 3.3 and 3.4.

3.3.3 Proof of Correctness

To prove the correctness of HDP Code, we take the case of one stripe for example here. The reconstruction of multiple stripes is just a matter of scale and similar to the reconstruction of one stripe. In a stripe, we have the following lemma and theorem,

Lemma 3.1. *We can find a sequence of a two-integer tuple (T_k, T'_k) where*

$$T_k = \left\langle p - 2 + \frac{k+1+\frac{1+(-1)^k}{2}}{2}(f_2 - f_1) \right\rangle_{p-1},$$
$$T'_k = \frac{1+(-1)^k}{2} f_1 + \frac{1+(-1)^{k+1}}{2} f_2 \quad (k = 0, 1, \cdots, 2p-3)$$

with $0 < f_2 - f_1 < p - 1$, all two-integer tuples $(0, f_1)$, $(0, f_2)$, $\cdots$, $(p-2, f_1)$, $(p-2, f_2)$ occur exactly once in the sequence. The similar proof of this lemma can be found in many papers on RAID-6 codes [8, 18, 86, 45].

Theorem 3.1. *A $p-1$-row-$p-1$-column stripe constructed according to the formal description of HDP Code can be reconstructed under concurrent failures from any two columns.*

Proof. The two failed columns are denoted as f_1 and f_2, where $0 < f_1 < f_2 < p$.

In the construction of HDP Code, any two horizontal-diagonal parity elements cannot be placed in the same row, as well for any two anti-diagonal parity elements. For any two concurrently failed columns f_1 and f_2, based on the layout of HDP Code, two data elements $C_{p-1-f_2+f_1,f_1}$ and $C_{f_2-f_1-1,f_2}$ can be reconstructed since the corresponding anti-diagonal parity element does not appear in the other failed column.

For the failed columns f_1 and f_2, if a data element C_{i,f_2} on column f_2 can be reconstructed, we can reconstruct the missing data element $C_{\langle i+f_1-f_2\rangle_p,f_1}$ on the same anti-diagonal parity chain if its corresponding anti-diagonal parity elements exist. Similarly, a data element C_{i,f_1} in column f_1 can be reconstructed, we can reconstruct the missing data element $C_{\langle i+f_2-f_1\rangle_p,f_2}$ on the same anti-diagonal parity chain if its anti-diagonal parity element parity element exist. Let us consider the construction process from data element $C_{f_2-f_1-1,f_2}$ on the f_2th column to the corresponding endpoint (element C_{f_1,f_1} on the f_1th

column). In this reconstruction process, all data elements can be reconstructed and the reconstruction sequence is based on the sequence of the two-integer tuple in Lemma 3.1. Similarly, without any missing parity elements, we may start the reconstruction process from data element C_{f_2-1,f_1} on the f_1th column to the corresponding endpoint (element C_{f_2,f_2} on the f_2th column).

In conclusion, HDP Code can be reconstructed under concurrent failures from any two columns. □

3.3.4 Reconstruction Process

We first consider how to recover a missing data element since any missing parity element can be recovered based on Equations 3.3 and 3.4. If the horizontal-diagonal parity element and the related $p-2$ data elements exist, we can recover the missing data element (assuming it's C_{i,f_1} in column f_1 and $0 \leq f_1 \leq p-2$) using the following equation,

$$C_{i,f_1} = \sum_{j=0}^{p-2} C_{i,j} \quad (j \neq f_1) \tag{3.5}$$

If there exists an anti-diagonal parity element and its $p-3$ data elements, to recover the data element (C_{i,f_1}), first we should recover the corresponding anti-diagonal parity element. Assume it is in row r and this anti-diagonal parity element can be represented by $C_{r,p-2-r}$ based on Equation 3.4, then we have:

$$i = \langle 2r + f_1 + 2 \rangle_p \tag{3.6}$$

So r can be calculated by (k is an arbitrary positive integer):

$$r = \begin{cases} (i - f_1 + p - 2)/2 & (i - f_1 = \pm 2k + 1) \\ (i - f_1 + 2p - 2)/2 & (i - f_1 = -2k) \\ (i - f_1 - 2)/2 & (i - f_1 = 2k) \end{cases} \tag{3.7}$$

According to Equation 3.4, the missing data element can be recovered,

$$\begin{aligned} C_{i,f_1} &= C_{r,p-2-r} \oplus \sum_{j=0}^{p-2} C_{\langle 2i+j+2 \rangle_p, j} \\ &(j \neq f_1 \quad and \quad j \neq \langle p-3-2i \rangle_p) \end{aligned} \tag{3.8}$$

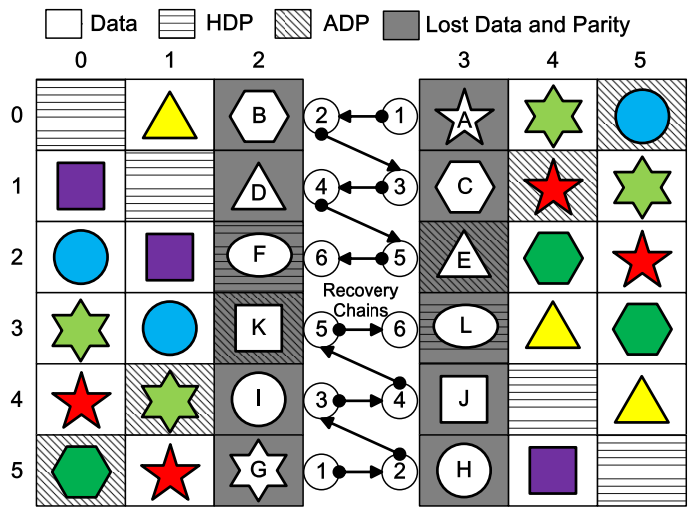


Figure 3.4: Reconstruction by two recovery chains.

In this figure, there are double failures in columns 2 and 3. To recover the lost columns, first we identify the two starting points of recovery chain: data elements A and G. Second we reconstruct data elements according to the corresponding recovery chains until they reach the endpoints (data elements F and L).The orders to recover data elements are: one is A→B→C→D→E→F, the other is G→H→I→J→K→L.

Based on Equations 3.5 to 3.8, we can easily recover the elements upon single disk failure. If two disks failed (for example, column f_1 and column f_2, $0 \leq f_1 < f_2 \leq p-2$), based on Theorem 3.1, we have our reconstruction algorithm of HDP Code shown in Algorithm 3.1.

Algorithm 3.1: Reconstruction Algorithm of HDP Code

Step 1: Identify the double failure columns: f_1 and f_2 ($f_1 < f_2$).
Step 2: Start reconstruction process and recover the lost data and parity elements.
switch $0 \leq f_1 < f_2 \leq p-2$ **do**
  Step 2-A: Compute two starting points ($C_{p-1-f_2+f_1,f_1}$ and $C_{f_2-f_1-1,f_2}$) of the recovery chains based on Equations 3.6 to 3.8.
  Step 2-B: Recover the lost data elements in the two recovery chains.
  Two cases start synchronously:
  case *starting point is* $C_{p-1-f_2+f_1,f_1}$ **repeat**
    (1) Compute the next lost data element (in column f_2) in the recovery chain based on Equation 3.5;
    (2) Then compute the next lost data element (in column f_1) in the recovery chain based on Equations3.6 to 3.8.
  until *at the endpoint of the recovery chain.*
  case *starting point is* $C_{f_2-f_1-1,f_2}$ **repeat**
    (1) Compute the next lost data element (in column f_2) in the recovery chain based on Equation 3.5;
    (2) Then compute the next lost data element (in column f_1) in the recovery chain based on Equations 3.6 to 3.8.
  until *at the endpoint of the recovery chain.*

In Algorithm 3.1, we notice that the two recovery chains of HDP Code can be recovered synchronously, and this process will be discussed in detail in Section 3.5.2.

3.3.5 Property of HDP Code

From the proof of HDP Code's correctness, HDP Code is essentially an MDS code, which has optimal storage efficiency [86, 15, 45].

3.4 Load Balancing Analysis

In this section, we evaluate HDP to demonstrate its effectiveness on load balancing.

3.4.1 Evaluation Methodology

We compare HDP Code with following popular codes in typical scenarios (when $p = 5$ and $p = 7$):

- **Codes for $p - 1$ disks**: HDP Code;
- **Codes for p disks**: P-Code [45];
- **Codes for $p + 1$ disks**: RDP code [18];
- **Codes for $p + 2$ disks**: EVENODD code [8].

For each coding method, we analyze a trace as shown in Figure 3.5. We can see that most of the write requests are 4KB and 8 KB in Microsoft Exchange application. Typically a stripe size is 256KB [20] and a data block size is 4KB, so single write and partial stripe write to two continuous data elements are dominant and have significant impacts on the performance of disk array.

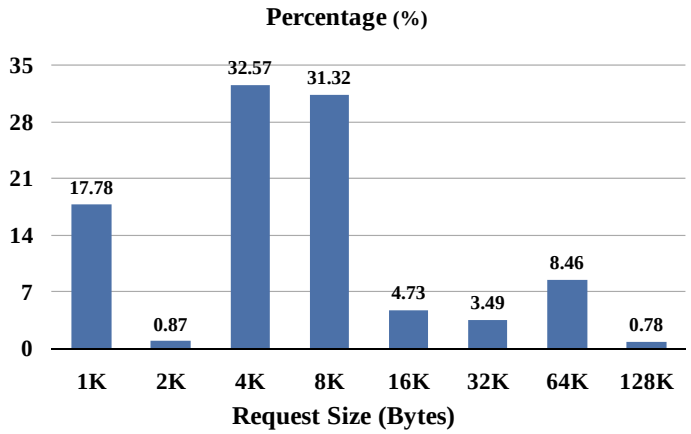


Figure 3.5: Write request distribution in *Microsoft Exchange* trace (most write requests are 4KB and 8KB).

Based on the analysis of exchange trace, we select various types of read/write requests to evaluate various codes as follows:

- **Read (R)**: read only;
- **Single Write (SW)**: only single write request without any read or partial write to continuous data elements;
- **Continuous Write (CW)**: only write request to two continuous data elements without any read or single write;
- **Mixed Read/Write (MIX)**: mixed above three types. For example, "RSW" means mixed read and single write requests, "50R50SW" means 50% read requests and 50% single write requests.

To show the status of I/O distribution among different codes, we envision an ideal sequence to read/write requests in a stripe as follows,

For each data element, it is treated as the beginning read/written element at least once. If there is no data element at the end of a stripe, the data element at the beginning of the stripe will be written.

Based on the description of ideal sequence, for HDP Code shown in Figure 3.3 (which has $(p-3)*(p-1)$ total data elements in a stripe), the ideal sequence of CW requests to $(p-3)*(p-1)$ data elements is: $C_{0,1}C_{0,2}$, $C_{0,2}C_{0,3}$, $C_{0,3}C_{0,4}$, and so on, $\cdots\cdots$, the last partial stripe write in this sequence is $C_{p-2,p-3}C_{0,0}$.

We also select two access patterns combined with different types of read/write requests:

- **Uniform access**. Each request occurs only once, so for read requests, each data element is read once; for write requests, each data element is written once; for continuous write requests, each data element is written twice.

Table 3.2: 50 Random Integer Numbers

221	811	706	753	34	862	353	428	99	502
32	800	969	346	889	335	361	209	609	11
18	76	136	303	175	71	427	143	870	855
706	297	50	824	324	212	404	199	11	56
822	301	430	558	954	100	884	410	604	253

- **Random access**. Since the number of total data elements in a stripe is less than 49 when $p = 5$ and $p = 7$, we use 50 random numbers (ranging from 1 to 1000) generated by a random integer generator [66] as the frequencies of partial stripe writes in the sequence one after another. These 50 random numbers are shown in Table 3.2. For example, in HDP Code, the first number in the table,"221" is used as the frequency of a CW request to elements $C_{0,1}C_{0,2}$.

We define $O_j(C_{i,j})$ as the number of I/O operations in column j, which is caused by a request to data element(s) with beginning element $C_{i,j}$. And we use $O(j)$ to delegate the total number of I/O operations of requests to column j in a stripe, which can be calculated by,

$$O(j) = \sum O_j(C_{i,j}) \tag{3.9}$$

We use O_{max} and O_{min} to delegate maximum/minimum number of I/O operations among all columns, obviously,

$$O_{max} = \max O(j), \quad O_{min} = \min O(j) \tag{3.10}$$

We evaluate HDP Code and other codes in terms of the following metrics. We define "**metric of load balancing** (L)" to evaluate the ratio between the columns with the highest

I/O O_{max} and the lowest I/O O_{min}. The smaller value of L is, the better load balancing is. L can be calculated by,

$$L = \frac{O_{max}}{O_{min}} \tag{3.11}$$

For uniform access in HDP Code, O_{max} is equal to O_{min}. According to Equation 3.11,

$$L = \frac{O_{max}}{O_{min}} = 1 \tag{3.12}$$

3.4.2 Numerical Results

In this subsection, we give the numerical results of HDP Code compared to other typical codes using above metrics.

First, we calculate the metric of load balancing (L values) for different codes with various p shown in Figure 3.6 and Figure 3.7. In the following figures, because there is no I/O in parity disks, the minimum I/O $O_{min} = 0$ thus $L = \infty$. The results show that HDP Code has the lowest L value and thus the best balanced I/O compared to the other codes.

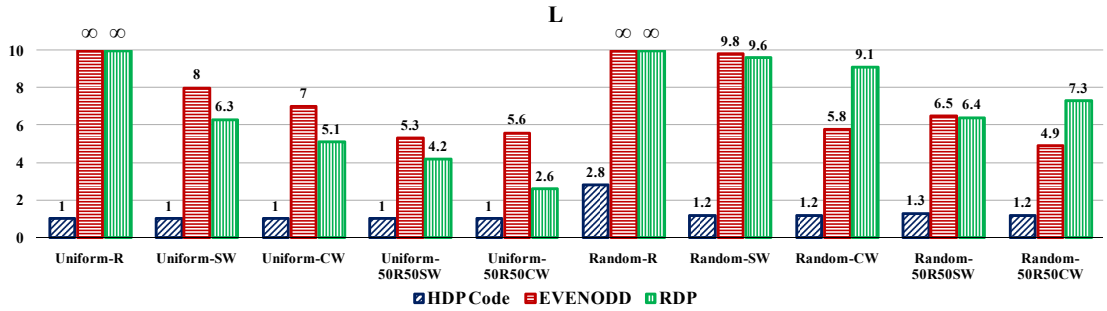


Figure 3.6: Metric of load balancing L under various codes ($p = 5$).

Second, to discover the trend of unbalanced I/O in horizontal codes, especially for uniform write requests, we also calculate the L value in RDP, EVENODD and HDP.

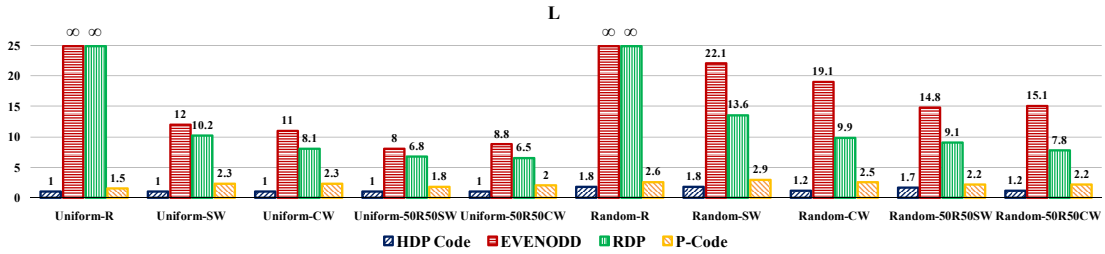


Figure 3.7: Metric of load balancing L under various codes ($p = 7$).

Table 3.3: Different L Value in Horizontal Codes

Code	Uniform SW requests	Uniform CW requests
RDP	$2p - 4 + \frac{1}{p-1}$	$\frac{3p}{2} - \frac{5}{2} + \frac{1}{2p-2}$
EVENODD	$2p - 2$	$2p - 3$
HDP	1	1

For HDP Code, which is well balanced as calculated in Equation 3.12 for any uniform request. For RDP and EVENODD codes, based on their layout, the maximum number of I/O operations appears in their diagonal parity disk and the minimum number of I/O operations appears in their data disk. By this method, we can get different L according to various values of p. For example, for uniform SW requests in RDP code,

$$L = \frac{O_{max}}{O_{min}} = \frac{2(p-1)^2 + 2(p-2)^2}{2(p-1)} = 2p - 4 + \frac{1}{p-1} \tag{3.13}$$

We summarize the results in Table 3.3 and give the trend of L in horizontal codes with different p values in Figure 3.8 and 3.9. With the increasing number of disks (p becomes larger), RDP and EVENODD suffer extremely unbalanced I/O while HDP code gets well balanced I/O in the write-intensive environment.

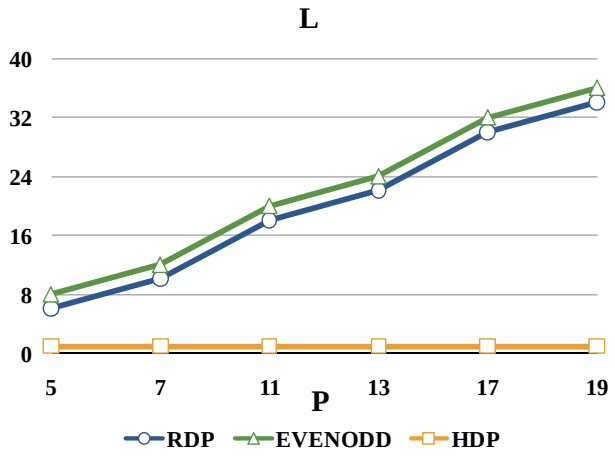


Figure 3.8: Metric of load balancing L under uniform SW requests with different p in various horizontal codes.

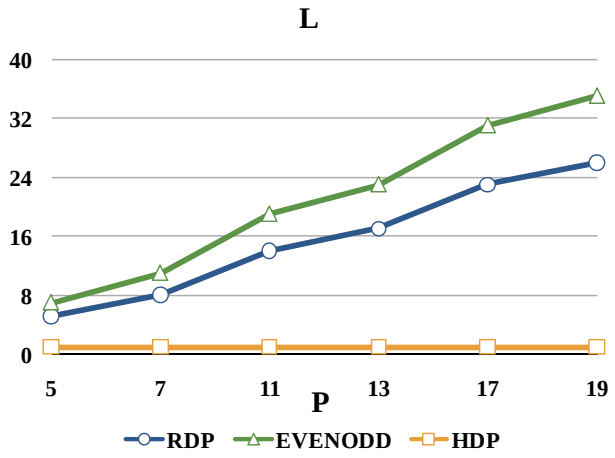


Figure 3.9: Metric of load balancing L under uniform CW requests with different p in various horizontal codes.

3.5 Reliability Analysis

By using special horizontal-diagonal and anti-diagonal parities, HDP Code also provides high reliability in terms of fast recovery on single disk and parallel recovery on double disks.

3.5.1 Fast Recovery on Single Disk

As discussed in Section 1.3.3, some MDS codes like RDP can be optimized to reduce the recovery time when single disk fails. This approach also can be used for HDP Code to get higher reliability. For example, as shown in Figure 3.10, when column 0 fails, not all elements need to be read for recovery. Because there are some elements like $C_{0,4}$ can be shared to recover two failed elements in different parities. By this approach, when $p = 7$, HDP Code reduces up to 37% read operations and 26% total I/O per stripe to recover single disk, which can decrease the recovery time thus increase the reliability of the disk array.

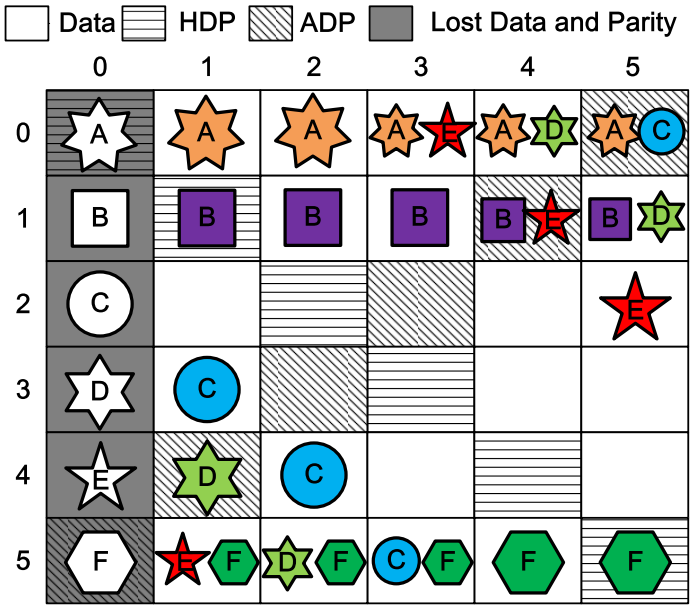


Figure 3.10: Single disk recovery in HDP Code when column 0 fails.

In this figure, the corresponding data or parity elements to recover the failed elements are signed with the same character, e.g., element $C_{0,4}$ signed with "AD" is used to recover the elements A and D.

We summarize the reduced read I/O and total I/O in different codes in Table 3.4. We can see that HDP achieves highest gain on reduced I/O compared to RDP and X-Code. Based on the layout of HDP Code, we also keep well balanced read I/O when one disk fails. For

Table 3.4: Reduced I/O of Single Disk failure in Different Codes

Code	Reduced Read I/O	Reduced Total I/O
Optimized RDP ($p = 5$)	25%	16.7%
Optimized X-Code ($p = 5$)	13%	12%
Optimized HDP ($p = 5$)	33%	20%
Optimized RDP ($p = 7$)	25%	19%
Optimized X-Code ($p = 7$)	19%	14%
Optimized HDP ($p = 7$)	37%	26%

example, as shown in Figure 3.11, the remaining disks all 4 read operations, which are well balanced.

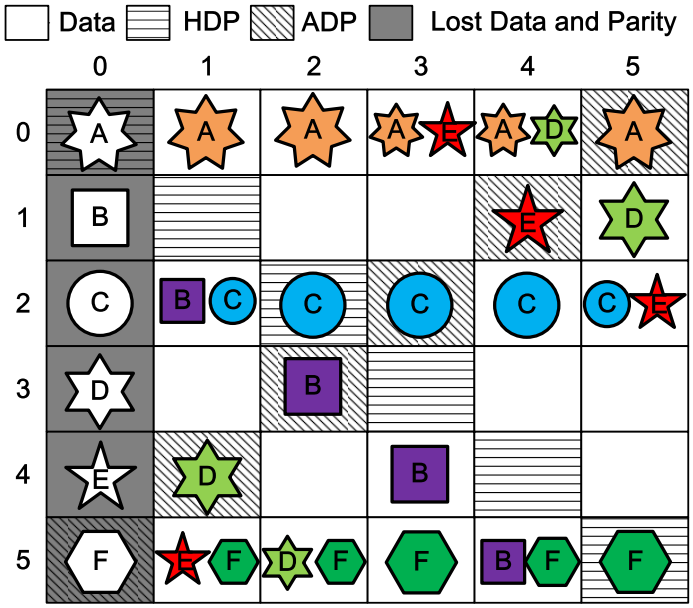


Figure 3.11: Single disk recovery in HDP Code when column 0 fails.

In this figure, each remain column has 4 read operations to keep load balancing well. Data elements A, C and F are recovered by their horizontal-diagonal parity while B, D and E are recovered by their anti-diagonal parity.

3.5.2 Parallel Recovery on Double Disks

According to Figure 3.4 and Algorithm 3.1, HDP Code can be recovered by two recovery chains all the time. It means that HDP Code can be reconstructed in parallel when any

Table 3.5: Recovery Time of Any Two Disk Failures in HDP Code ($p = 5$)

failed columns						total recovered elements
$0, 1$	$0, 2$	$0, 3$	$1, 2$	$1, 3$	$2, 3$	
$6R_t$	$6R_t$	$4R_t$	$4R_t$	$6R_t$	$6R_t$	8*6=48

two disks fail, which can reduce the recovery time of double disks failure thus improve the reliability of the disk array. To efficiently evaluate the effect of parallel recovery, we assume that R_t is the average time to recover a data/parity element, then we can get the parallel recovery time of any two disks as shown in Table 3.5 when $p = 5$. Suppose $N_{avg.}$ delegates the average number of elements can be recovered in a time interval R_t, the larger of $N_{avg.}$ value is, the better parallel recovery on double disks and the lower recovery time will be. And we can calculate $N_{avg.}$ value by the results shown in Table 3.5,

$$N_{avg.} = \frac{48}{6 + 6 + 4 + 4 + 6 + 6} = 1.5 \tag{3.14}$$

We also get the $N_{avg.} = 1.43$ when $p = 7$ in our HDP Code. It means that HDP can recover 50% or 43% more elements during the same time interval. It also shows that our HDP Code reduces 33% or 31% total recovery time compared to most RAID-6 codes with single recovery chain as RDP [18], Cyclic [15] and P-Code [45].

3.6 Summary

In this chapter, we propose the Horizontal-Diagonal Parity Code (HDP Code) to optimize the I/O load balancing for RAID-6 by taking advantage of both horizontal and diagonal/anti-diagonal parities in MDS codes. HDP Code is a solution for an array of

$p - 1$ disks, where p is a prime number. The parities in HDP Code include horizontal-diagonal parity and anti-diagonal parity, where all parity elements are distributed among disks in the array to achieve well balanced I/O. Our mathematic analysis shows that HDP Code achieves the best load balancing and high reliability compared to other MDS codes.

Chapter 4

Stripe-based Data Migration (SDM) Scheme

4.1 Introduction

In recent years, with the development of cloud computing, scalability becomes an important issue [2] which is urgently demanded by RAID systems due to following reasons,

1. By extending more disks, a disk array provides higher I/O throughput and larger capacity [87]. It not only satisfies the sharp increasing on user data in various online applications [27], but also avoids the extremely high downtime cost [56].

2. RAID-based architectures are widely used for clusters and large scale storage systems, where scalability plays a significant role [43, 70].

3. The storage efficiency can be improved by adding more disks into an existing disk array which also decreases the cost of the storage system.

However, existing solutions in disk arrays [29, 55, 88, 28, 89, 37, 90] are not suitable for RAID-6 scaling process to add disks to an existing disk array. Researchers face a challenge to find an effective solution to scale RAID-6 systems based on MDS codes efficiently. First, existing approaches are proposed for general case in RAID-0 or RAID-5 [29, 55, 88, 28, 89, 37, 90], which cannot adopt various coding methods in RAID-6. For example, RDP and P-Code have different layouts of data and parity. Thus the scaling scheme should be designed according to the characteristic of RDP or P-Code, respectively. Second, typical scaling schemes are based on round-robin order [29, 55, 88, 89], which are not suitable for RAID-6 due to high overhead on parity migration, modification and computation. One of the reasons is that the parity layouts of RAID-6 codes are complex. Another reason is that the stripes are dramatically changed after scaling. For example, a movement on any data element may lead to up to eight additional I/O operations on its corresponding parity elements.

To address the above challenge, we propose a new **S**tripe-based **D**ata **M**igration (**SDM**) scheme to accelerate RAID-6 scaling. Different from existing approaches, SDM exploits the relationships between the stripe layouts before and after scaling to make scaling process efficiently.

The rest of this chapter continues as follows: Section 4.2 discusses the motivation. SDM scheme is described in detail in Section 4.3. Section 4.4 gives the quantitative analysis on scalability. Finally we conclude the chapter in Section 4.5.

Table 4.1: Summary on Various Fast Scaling Approaches

Name	Features in Section 1.3.4				Used in RAID-6?
	Feature 1	Feature 2	Feature 3	Feature 4	
RR	√	×	√	×	conditionally
Semi-RR	×	×	√	×	conditionally
ALV	√	×	√	×	×
MDM	×	√	√	√	×
FastScale	√	√	√	√	×
SDM	√	√	√	√	√

4.2 Motivation

We summarize the existing fast scaling approaches in Table 4.1. Although these fast scaling approaches offer some nice features for RAID-0 or RAID-5, it is not suitable for RAID-6 due to the special coding methods, of which MDS codes are popular. Figures 1.4 and 1.6 show the parity layout of RDP [18] and P-Code [45], where there are several problems regarding scalability. First, existing scaling approaches are difficult to be applied in these codes. If we extend a RAID-6 disk array based on RDP from 6 to 8 disks, any new data cannot be migrated into the dedicated parity disks (columns 6 and 7). On the other hand, if we select P-Code, we needn't care about the dedicated parity disks. Second, few strategies on reducing parity modification are involved in existing methods. Actually, for keeping parity consistency, too many parity elements have to be recalculated and modified because of the movements on their corresponding data elements in RAID-6. It causes high computation cost, modification overhead, and an unbalanced migration I/O.

In summary, existing scaling approaches are insufficient to satisfy the desired four features listed in Section 1.3.4, which motivates us to propose a new approach on RAID-6 scaling.

4.3 SDM Scheme

In this section, an **S**tripe-based **D**ata **M**igration (SDM) scheme is designed to accelerate the RAID-6 scaling. The purpose of SDM is to minimize the parity migration, modification and computation according to a global point view on single/multiple stripe(s), not limited to a migration on single data/parity element as Round-Robin [29, 55, 88].

To clearly illustrate the stripes before/after scaling, we define four types of stripes as follows,

- **Old Used Stripe (OUS)**: A used stripe before scaling.
- **Old Empty Stripe (OES)**: An empty stripe before scaling.
- **New Used Stripe (NUS)**: A used stripe after scaling.
- **New Empty Stripe (NES)**: An empty stripe after scaling.

In our SDM scheme, an OUS/OES corresponds to a NUS/NES with the same stripe ID, respectively. SDM scheme takes the following steps,

1. Priority Definition: Define the different priorities for the movements on single data/parity.

2. Layout Comparison: Compare the layouts before and after scaling and find a cost-effective way to change the layout of a stripe. The data/parity movements in this step should have the highest priority.

3. Load Balancing Check: According to the data distribution in the stripes (after Step 2), select a small portion of stripes to balance the workload. The data/parity movements in this step can have acceptable overhead.

Table 4.2: Priorities of Data/Parity Migration in SDM (For A Movement on Single Data/Parity Element)

Priority	Number of modified Parities (for parity consistency)	Overhead of Data/Parity Migration & Modification	Parity Computation Cost
1	none	2 I/Os	none
2	one	4 I/Os	1 XOR
3	two	6 I/Os	2 XORs
4	three or more	$\geq$ 8 I/Os	4 XORs

4. Data Migration: Based on the migration schemes in Steps 2 and 3, start data migration for each stripe.

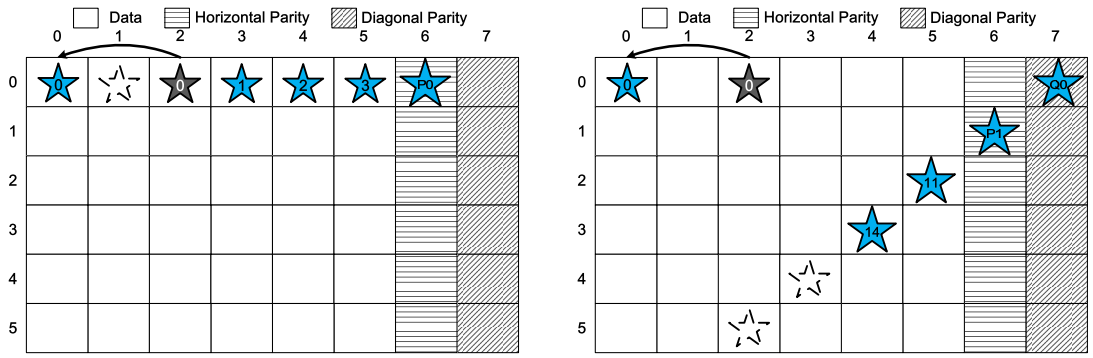
Typically, without any special instructions, a data/parity block (in logical address view) corresponds to a data/parity element (in parity layout view) in a stripe. Due to the difference between horizontal and vertical codes, we design various scaling schemes and discuss them separately. In this section, we use RDP [18] (a typical horizontal code) and P-Code [45] (a typical vertical code) as examples to show how SDM works in RAID-6, scaling from 6 to 8 disks and from 6 to 7 disks, respectively. Their corresponding parity layouts are shown in Figures 1.4 and 1.6.

4.3.1 Priority Definition of Data/Parity Movements

Due to complex parity layouts in RAID-6, several scenarios on data movements should be considered. Due to this reason, we define the priorities of data/parity migration in SDM according to the number of modified parities as summarized in Table 4.2. Higher priority means lower overhead on total IOs and parity computation.

According to the various priorities in Table 4.2, we can measure whether a movement is efficient or not. For example, as shown in Figure 4.1, Block 0 is migrated from disk 2

to disk 0, and other blocks are retained without any movement. From horizontal parity's point of view (shown in Figure 4.1(a)), Block 0 also stays in the original horizontal parity chain and the corresponding parity $P0$ needn't be changed. On the other hand, considering the aspect of diagonal parity (shown in Figure 4.1(b)), Block 0 also shares the same parity chain with other blocks (e.g., Blocks 11, 14, $P1$ and $Q0$), and the corresponding parity element could be retained. Therefore, the movement of Block 0 doesn't change any parity and has the highest priority (Priority 1) in the scaling process.



(a) Horizontal parity view.

(b) Diagonal parity view.

Figure 4.1: Data movement of Block 0 in Figure 1.10.

4.3.2 Layout Comparison

Compared to the current and future parity layouts, we propose different rules for horizontal and vertical codes,

Rules for Horizontal Codes

- *(Parity Disks Labeling)* Original parity disks are retained and the extending disk(s) are used for data disk(s).

- *(Extended Disk(s) Labeling)* If m disk(s) are added into a disk array, the new disk(s) are labeled as the m middle column(s) (in the middle of all data columns).

- *(Rows Labeling)* If an Old Used Stripe (OUS) contains n_r rows, these rows are labeled as the first n_r rows in the New Used Stripe (NUS).

- *(Special parity handling)* If horizontal parities take part in the calculation of diagonal/anti-diagonal parities, the priority of data/parity movement in the horizontal chain is higher than that in diagonal/anti-diagonal parity chain. Conversely, if diagonal/anti-diagonal parities take part in the calculation of horizontal parities, the priority of data/parity movement in the diagonal/anti-diagonal chain is higher than that in horizontal chain.

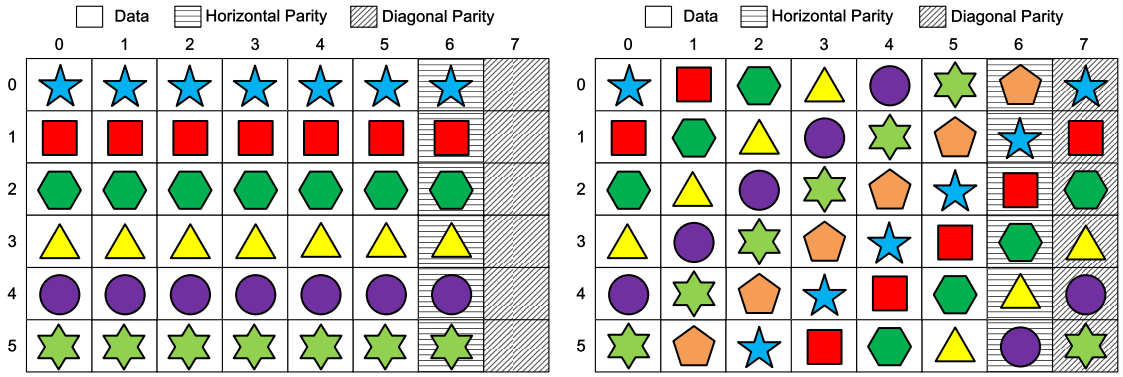
- *(Data/Parity Migration)* Select proper data/parity movements with the highest priority.

For example, if we want to scale a RAID-6 array using RDP from 6 to 8 disks, compared to the layouts in Figures 1.4 and 4.2, we have the following strategies according to the above rules,

1.(Parity Disks Labeling) Columns 4 and 5 in Figure 1.4 are retained as parity columns, and the column IDs are changed to columns 6 and 7 in Figure 4.2.

2.(Extended Disk(s) Labeling) Two extended disks are regarded as data disks and labeled as columns 2 and 3, causing the lowest overhead as shown in Table 4.3.

3.(Rows Labeling) The row IDs in each Old Used Stripe (OUS) are retained. By extending more disks, the size of stripes becomes larger and contains several new rows, which are known as "phantom" rows [62] in the scaling process.



(a) Horizontal parity coding. (b) Diagonal parity coding.

Figure 4.2: RDP code for $p + 1$ disks ($p = 7$, $n = 8$).

Table 4.3: Different overhead of Extended Disk(s) Labelling in SDM

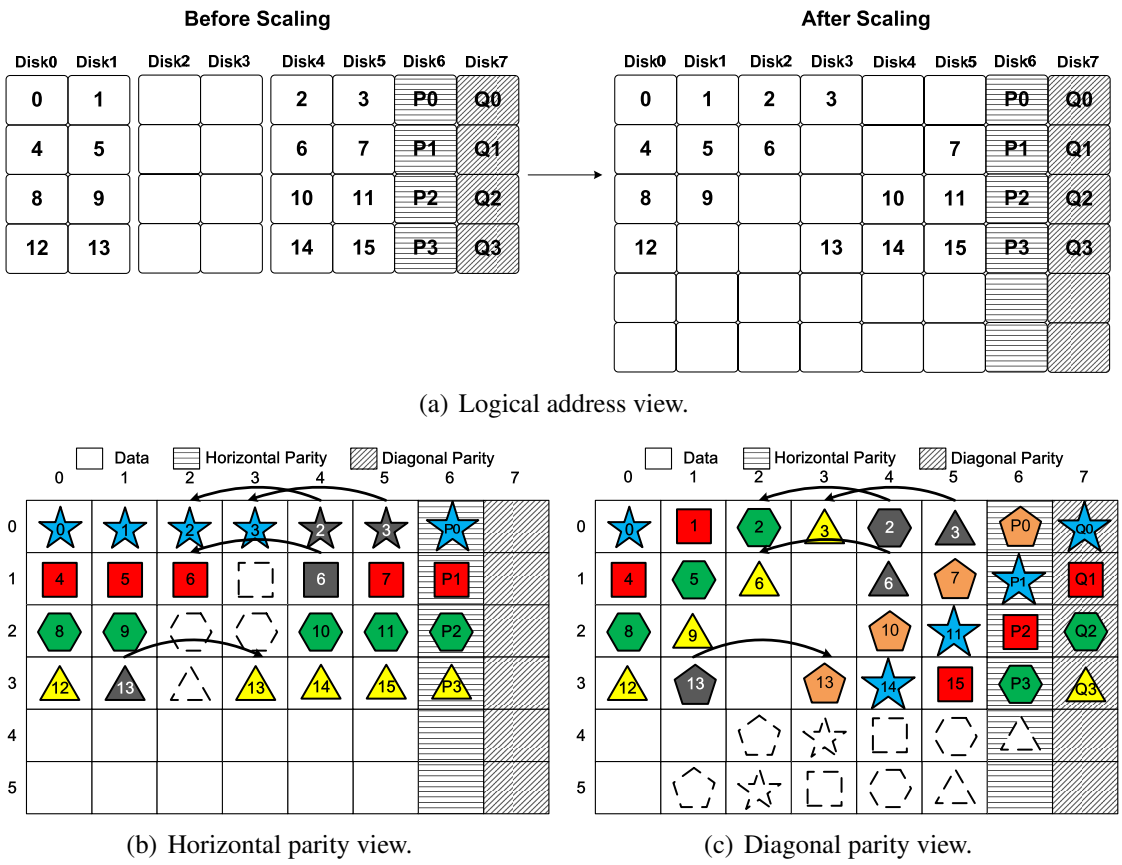
Extended disk(s) labeling	Minimal number of moved data/parity	Migration Cost
columns 0 and 1	10	20 I/Os
columns 1 and 2	6	12 I/Os
columns 2 and 3	4	8 I/Os
columns 3 and 4	4	8 I/Os
columns 4 and 5	6	12 I/Os
other cases	≥ 6	≥ 12 I/Os

4.(Special parity handling) The priority of data/parity movement in the horizontal chain is higher than that in diagonal/anti-diagonal parity chain.

5.(Data/Parity Migration) Blocks 2, 6, 3 and 13 are selected to migrate as shown in Figure 4.3, and all data blocks also share the same parities with the new parity layout. Therefore, all parity blocks are retained and these movements have the highest priority.

Rules for Vertical Codes

- *(Original Disks Labeling)* Original disk IDs are retained and the extended disks are used for data disks.



(a) Logical address view.

(b) Horizontal parity view.

(c) Diagonal parity view.

Figure 4.3: Data/parity migration in Layout Comparison Step (RDP code, scaling from 6 to 8 disks).

- *(Extended Disk(s) Labeling)* By extending m disk(s) into a disk array, the new disks are labeled as the last m columns.

- *(Rows Labeling)* If an Old Used Stripe (OUS) contains n_r data rows, which are labeled as the same row ID in the New Used Stripe (NUS). The dedicated parity rows are labeled according to the new layout.

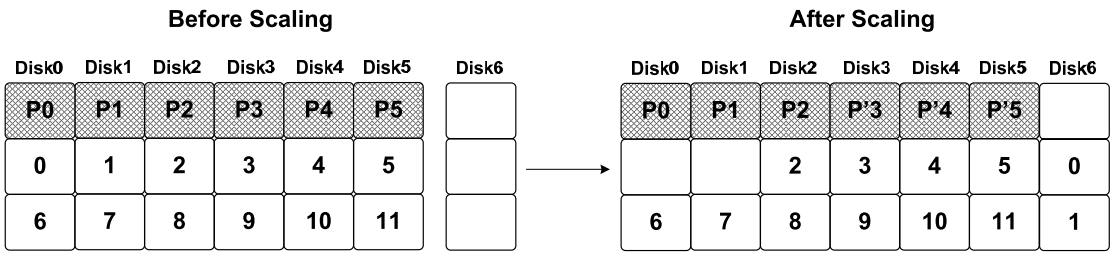
- *(Data/Parity Migration)* Select proper data/parity movements which have the highest priority.

For example, if we want to scale a RAID-6 array using P-Code from 6 to 7 disks, compared to the layouts in Figure 1.6, we have the following strategies according to the above rules,

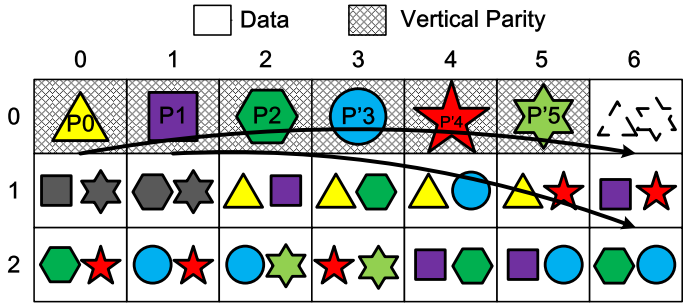
1.(Disks Labeling) As shown in Figure 4.4, original column IDs are retained. The new disk is labeled as Column 6.

2.(Rows Labeling) The row IDs in each Old Used Stripes (OUSs) are retained.

3.(Data/Parity Migration) Blocks 0 and 1 are selected to migrate as shown in Figure 4.4, which have the highest priority and three parities ($P3$, $P4$ and $P5$) are modified in each stripe.



(a) Logical address view.



(b) Vertical parity view.

Figure 4.4: Data/parity migration in Layout Comparison Step (P-Code, scaling from 6 to 7 disks).

Table 4.4: Data Distribution of RAID-6 Scaling by Using SDM Scheme (RDP code, Scaling from 6 to 8 disks)

Stripe ID	Total Data Elements	Column ID					
		0	1	2	3	4	5
0	16	4	3	2	2	2	3
1	16	4	3	2	2	2	3
2	16	0	2	4	4	4	2
stripe set (three stripes)	48	8	8	8	8	8	8

Table 4.5: Data Distribution of RAID-6 Scaling by Using SDM Scheme (P-Code, Scaling from 6 to 7 disks)

Stripe ID	Total Data Elements	Column ID						
		0	1	2	3	4	5	6
0	12	1	1	2	2	2	2	2
1	12	2	2	1	1	2	2	2
2	12	2	2	2	2	1	1	2
3	12	1	1	2	2	2	2	2
4	12	2	2	1	1	2	2	2
5	12	2	2	2	2	1	1	2
6	12	2	2	2	2	2	2	0
stripe set (seven stripes)	84	12	12	12	12	12	12	12

4.3.3 Load Balancing Check in SDM

In the load balancing checking step, first we get the statistics of data distribution after layout comparison. For example, the data distribution in RDP and P-Code are shown in the second row (Stripe ID is 0) in the Tables 4.4 and 4.5. We notice that it is unbalanced distribution, where different column has various number of data elements.

For horizontal and vertical codes, we use different ways to get a uniform data distribution as follows,

- *(Typically for horizontal codes)* Most horizontal codes have unsymmetrical data and parity distribution, a small portion of stripes will be sacrificed with a little higher

migration cost (called "sacrificed stripes"). In the load balancing checking step, the portion of sacrificed stripes will be calculated and the migration scheme in these stripes will be proposed.

- *(Typically for vertical codes)* Most vertical codes have symmetrical data and parity distribution, the data elements in each stripes will be migrated alternately, and a small portion of stripes will be chosen without any movement (called "retained stripes").

To calculate the percentage of sacrificed/retained stripes, we define "**Stripe Set**" which includes n_s stripes with uniform data distribution. Suppose $n_{d'}$ is the total number of data disks after scaling, and n_e is the total number of data elements in a stripe before scaling. n_s can be computed by,

$$n_s = \frac{lcm\left\{n_{d'}, n_e\right\}}{n_e} \tag{4.1}$$

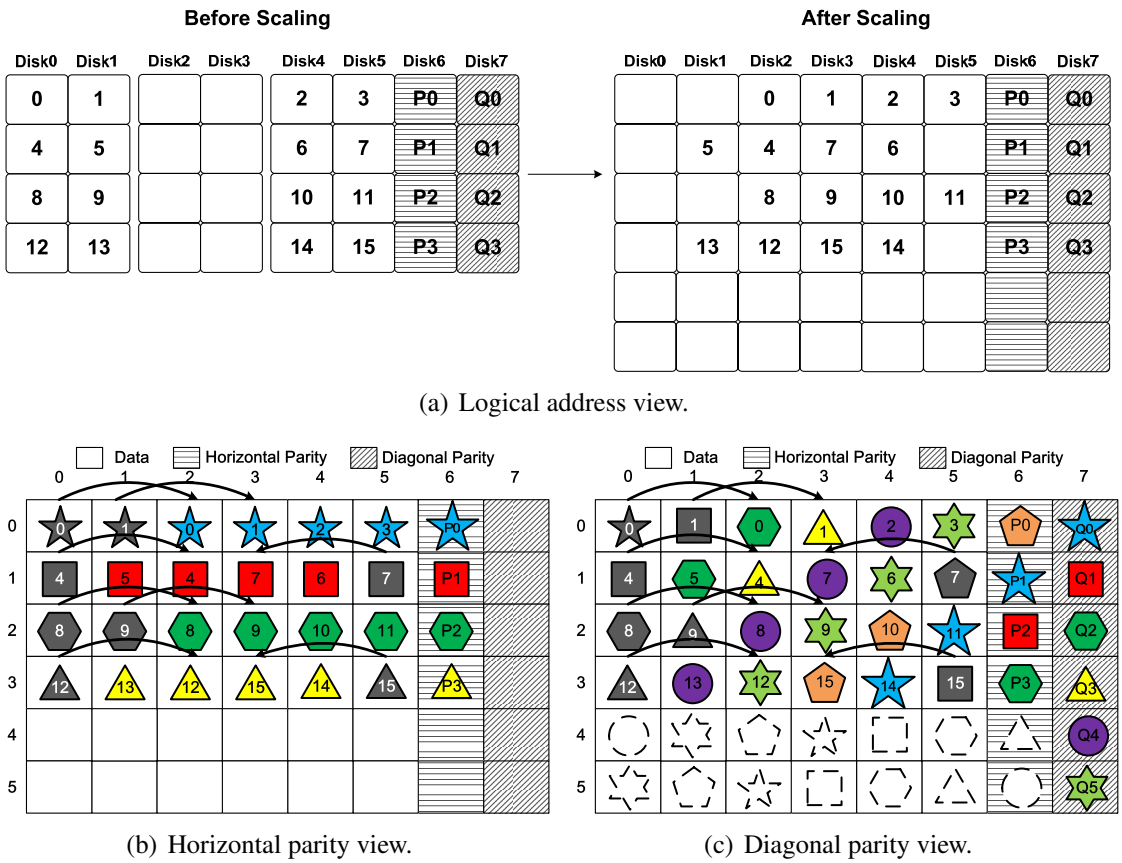
According to Equation4.1, in a stripe set, the total number of data elements in each column (denoted by n_{ec}) is,

$$n_{ec} = \frac{lcm\left\{n_{d'}, n_e\right\}}{n_{d'}} \tag{4.2}$$

Typically, a small portion of stripes in each stripe set will be selected as sacrificed or retained stripe and we can calculate the number of data elements in these stripes. For example, as shown in Table 4.4, each stripe set contains three stripes ($n_s = \frac{lcm\{16,6\}}{16} = 48/16 = 3$), where the last one is served as sacrificed stripe. In each stripe set, each column should contain 8 data elements ($n_{ec} = \frac{lcm\{16,6\}}{6} = 48/6 = 8$). So based on the number of data elements in each stripe after layout comparison, we can get the data distribution in the sacrificed stripe is $0, 2, 4, 4, 4, 2$ (e.g., the number of data elements in

column 0 is: $n_{ec} - 2 * 4 = 8 - 8 = 0$). Similarly, we also can get the stripe set size for P-Code is 7 ($n_s = \frac{lcm\{12,7\}}{12} = 84/12 = 7$) and each column should contain 12 data elements to maintain a uniform workload ($n_{ec} = \frac{lcm\{12,7\}}{7} = 84/7 = 12$), if data elements are migrated alternately as shown in Table 4.5, the last stripe is used for retained stripe.

Finally, we summarize the migration process in the load balancing checking step based on the priority level, shown in Figures 4.5 and 4.6. Due to space limit, in Figure 4.6, we only give the migration process of the second stripe in each stripe set (Stripe 1 in Table 4.5). Obviously, in Tables 4.4 and 4.5, each stripe set has an even data distribution.

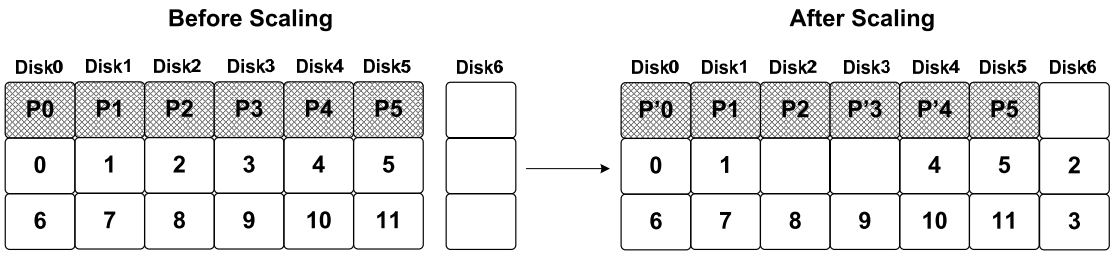


(a) Logical address view.

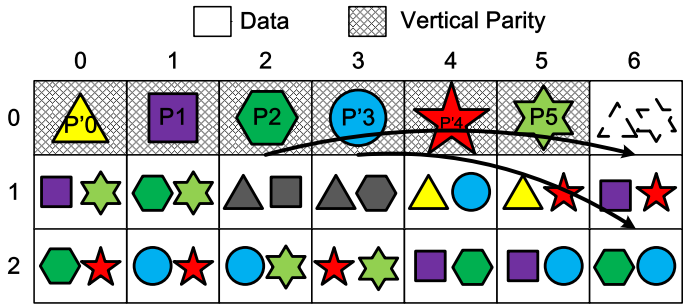
(b) Horizontal parity view.

(c) Diagonal parity view.

Figure 4.5: Data/parity migration in load balancing check step (RDP code, scaling from 6 to 8 disks).



(a) Logical address view.



(b) Vertical parity view.

Figure 4.6: Data/parity migration in load balancing check step (P-code, scaling from 6 to 7 disks).

4.3.4 Data Migration

After the steps of layout comparison and load balancing check, a comprehensive migration strategy can be derived and then the system can start the data migration process. We notice that Feature 2 can be satisfied and have the following theorem,

Theorem 4.1. *For any MDS code scaling from n disks to $n+m$ disks by using SDM scheme, the total number of migrated data blocks is $m * B/(m + n_d)$, where n_d is the number of data disks before scaling.*

Here we only demonstrate this theorem is correct for RDP scaling from 6 to 8 disks and P-Code scaling from 6 to 7 disks. The total number of migrated data blocks in the two examples are $B/3$ and $B/7$, respectively.

Proof. First we demonstrate the case for RDP scaling from 6 disks to 8 disks. In each stripe set, two stripes are migrated based on Figure 4.3, the remain one stripe is migrated according to Figure 4.5, so the total number of data movements in each stripe set is $4 * 2 + 8 = 16$. The total number of stripe set is $B/48$ and the total number of migrated data blocks is $16 * B/48 = B/3$. In this case, $m = 2$ and $n_d = 4$, the total number of migrated data blocks is $m * B/(m + n_d) = 4 * B/(4 + 2) = B/3$.

Second consider the other case for P-Code scaling from 6 disks to 7 disks. According to migration methods in Figures 4.4 and 4.6, the total number of data movements in each stripe set is 12. The total number of stripe set is $B/84$ and the total number of migrated data blocks is $12 * B/84 = B/7$. In this case, $m = 1$ and $n_d = 6$, the total number of migrated data blocks is also equal to $m * B/(m + n_d) = 1 * B/(6 + 1) = B/7$.

Thus the theorem is correct. □

4.3.5 Data Addressing Algorithm

The data addressing algorithms of the two examples on RDP and P-Code are shown in Algorithms 4.1 and 4.2. SDM scheme satisfies fast addressing feature in Section 1.3.4. For a disk array scaling from 6 to 8 then to 12 disks by using RDP code, our algorithms can be used multiple times by saving the initialization information.

4.3.6 Property of SDM

From the above discussion, we can see that SDM satisfies the desired features 1-3 of RAID-6 scaling defined in Section 1.3.4. SDM also satisfies the Feature 4: minimal modification and computation cost of the parity elements, which is discussed in detail in Sections 4.4.

Algorithm 4.1: Data Addressing Algorithm of RDP Scaling from n to $n+m$ disks Using SDM Scheme (where $n = p_1 + 1$, $n + m = p_2 + 1$, $p_1 < p_2$, p_1 and p_2 are prime numbers)

```
Get or calculate the S_id, i, j, n_s and n_ss value, then label the new disks with column IDs
from n − m/2 − 3 to n + m/2 − 4
S'_id = S_id;    /*Stripe ID unchanged*/
k = S_id mod n_s;
if 0 ≤ k ≤ n_s − n_ss − 1 (migrated stripes in layout comparison step) then
    if i + j ≤ p_1 − 1 then
        i' = i, j' = j;
    end
    else
        i' = i, j' = j + m.
    end
end
if n_s − n_ss ≤ k ≤ n_s − 1 (sacrificed stripes in load balancing checking step) then
    if (i = 1, 3, 5, ··· , p_1 − 1) && (j = 1, 3, 5, ··· , n + m − 3) then
        i' = i, j' = j;
    end
    else
        i' = i, j' = j + m.
    end
end
```

Algorithm 4.2: Data Addressing Algorithm of P-Code Scaling from n to $n+m$ disks Using SDM Scheme (where $n = p_1 - 1$, $n + m = p_2$, $p_1 \leq p_2$, p_1 and p_2 are prime numbers)

```
Get or calculate the S_id, i, j, n_s and n_rs value, then label the new disks with column IDs
from n to n + m − 1
S'_id = S_id;    /*Stripe ID unchanged*/
k = S_id mod n_s;
if 0 ≤ k ≤ n_s − n_rs − 1 (migrated stripes in layout comparison and load balancing
checking step) then
    if migrated data elements then
        distribute i' and j' based on round-robin order, where 0 ≤ i' ≤ (p_1 − 1)/2,
        n ≤ j' ≤ n + m − 1;
    end
    else
        i' = i, j' = j.
    end
end
if n_s − n_rs ≤ k ≤ n_s − 1 (retained stripes in load balancing checking step) then
    i' = i, j' = j.
end
```

4.4 Scalability Analysis

In this section, we evaluate the scalability of various MDS codes by using different approaches.

4.4.1 Evaluation Methodology

We compare SDM scheme to **Round-Robin (RR)** [29, 55, 88] and **Semi-RR** [28] approaches. ALV [89], MDM [37] and FastScale [90] cannot be used in RAID-6, so they are not evaluated.

We also propose an ideal fast scaling method as a baseline. The ideal case is based on the Feature 2 (Section 1.3.4) with minimal data movements to maintain a uniform workload in the enlarged new used stripe. We assume this case doesn't involve any parity migration, modification and computation as in RAID-0. Because no movement in dedicate parity disks (e.g., for RDP code), actually the number of ideal movements is $m * B/(m + n_d)$, where n_d is the number of data disks.

Several popular MDS codes in RAID-6 are selected for comparison,

1) Codes for $p - 1$ disks: P-Code (two variations are shown in Figure 1.6. [45] and HDP (Paper *III*, introduced in Chapter 3);

2) Codes for p disks: X-Code [86] and P-Code;

3) Codes for $p + 1$ disks: RDP code [18] and H-Code (Paper *IV*, introduced in Chapter 2);

4) Codes for $p + 2$ disks: EVENODD code [8].

Table 4.6: Overhead of RAID-6 Scaling by Using SDM Scheme (RDP code, Scaling from 6 disks to 8 disks)

Stripe ID	Number of Data & Parity Movements	Number of Modified Parities	Total I/Os	Number of XOR Calculations
0	4	none	8 I/Os	none
1	4	none	8 I/Os	none
2	8	16	48 I/Os	32 XORs
stripe set (three stripes)	16	16	64 I/Os	32 XORs

Suppose the total number of data blocks in a disk array is B, the total number of stripes in a disk array before scaling is S, we can derive the relationship between these two parameters. For example, for RDP code when $p = 5$, $B = 16S$; when $p = 7$, $B = 36S$.

We define **Data Migration Ratio** (R_d) is the ratio between the number of migrated data/parity blocks and the total number of data blocks. **Parity Modification Ratio** (R_p) delegates the ratio between the number of modified parity blocks and the total number of data blocks, which is caused by the data/parity migration.

For example, if we choose the case RDP code$(6, 2)$ using SDM scheme and according to the results are presented in Table 4.6,

$$\begin{cases} R_d = \frac{16S}{16S*3} = 33.3\% \\ R_p = \frac{16S}{16S*3} = 33.3\% \end{cases} \tag{4.3}$$

We also can get the results for P-Code$(6, 1)$ based on the digitals in Table 4.7,

$$\begin{cases} R_d = \frac{12S}{12S*7} = 14.3\% \\ R_p = \frac{24S}{12S*7} = 28.6\% \end{cases} \tag{4.4}$$

Table 4.7: Overhead of RAID-6 Scaling by Using SDM Scheme (P-Code, Scaling from 6 disks to 7 disks)

Stripe ID	Number of Data & Parity Movements	Number of Modified Parities	Total I/Os	Number of XOR Calculations
0	2	4	12 I/Os	8 XORs
1	2	4	12 I/Os	8 XORs
2	2	4	12 I/Os	8 XORs
3	2	4	12 I/Os	8 XORs
4	2	4	12 I/Os	8 XORs
5	2	4	12 I/Os	8 XORs
6	none	none	none	none
stripe set (seven stripes)	12	24	72 I/Os	48 XORs

In RAID-6 scaling, each data or parity migration only cost two I/O operations, and the modification cost of each parity is two I/Os as well. So based on the data migration ratio (R_d) and parity modification ratio (R_p), the total number of I/O operations is,

$$n_{io} = 2 * R_d * B + 2 * R_p * B \tag{4.5}$$

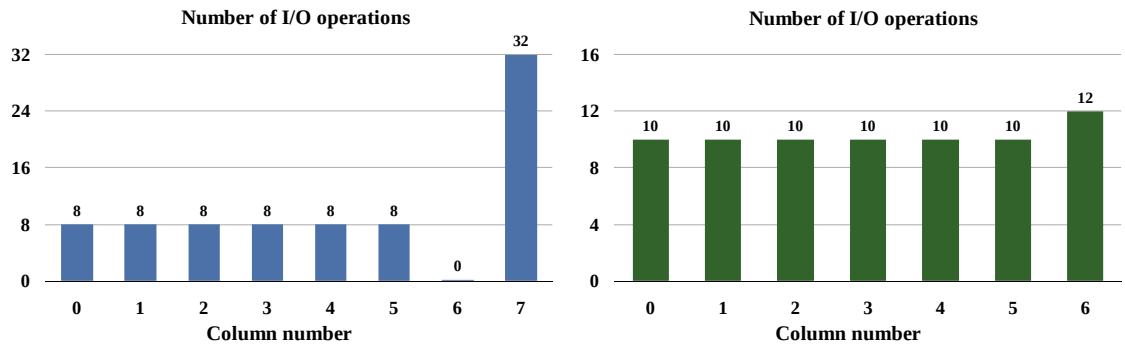
Based on Equation 4.5, by using SDM scheme, the total number of I/O operations for RDP Code$(6, 2)$ is $2 * B * 33.3\% + 2 * B * 33.3\% = 1.33B$, the total number of I/Os for P-Code$(6, 1)$ is $2 * B * 14.3\% + 2 * B * 28.6\% = 0.86B$.

If we ignore the computation time and assume the same time on a read or write request to a block (denoted by T_b), and suppose the migration I/O can be processed in parallel on each disk, the migration time T_m for RDP Code$(6, 2)$ using SDM scheme is (I/O distribution is shown in Figure 4.7(a) where column 7 has the highest I/O and longest migration time cost),

$$T_m = 32ST_b/3 = 2BT_b/3 \tag{4.6}$$

Similarly, based on Figure 4.7(b), the migration time of P-Code$(6, 1)$ using SDM scheme is (column 6 has the highest I/O and longest migration time cost),

$$T_m = 12ST_b/7 = BT_b/7 \tag{4.7}$$



(a) RDP Code (every three stripes, scaling from 6 to 8 disks).

(b) P-Code (every seven stripes, scaling from 6 to 7 disks).

Figure 4.7: I/O distribution in multiple stripes using SDM scheme.

4.4.2 Numerical Results

In this section, we give the numerical results of scalability using different scaling approaches and various coding methods. In the following Figures 4.8 to 4.13, a two-integer tuple (n, m) denotes the original number of disks and the extended number of disks. For example, RDP $(6, 2)$ means a RAID-6 scaling from 6 to $6 + 2$ disks using RDP code.

Data Distribution

Regarding to data distribution, we use the coefficient of variation as a metric to examine whether the distribution is even or not as other approaches [28, 90]. The coefficient of variation delegates the standard deviation (a percentage on average). The small value of

the coefficient of variation means highly uniform distribution. From the introduction in Section 4.2, Semi-RR suffers I/O load balancing problem, RR and SDM keep a uniform distribution.

The results are shown in Figure 4.8. We notice that semi-RR and SDM without load balancing checking causes extremely unbalanced I/O, which cannot satisfy Feature 1 (uniform distribution).

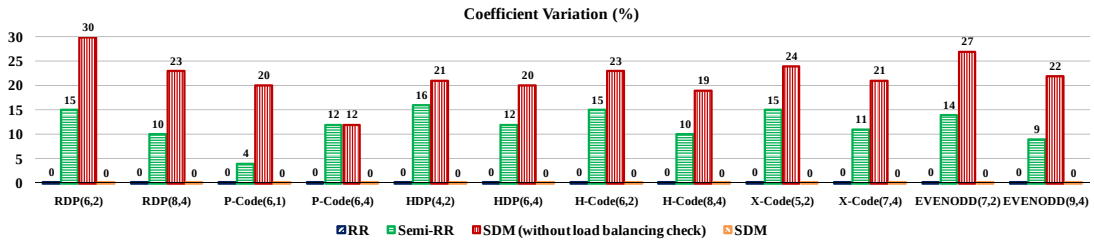


Figure 4.8: Comparison on data distribution under various RAID-6 scaling approaches.

Data Migration Ratio

Second, we calculate the data migration ratio (R_d) among various fast scaling approaches under different cases as shown in Figure 4.9. Our SDM scheme has the approximate migration ratio compared to Semi-RR and the ideal case in RAID-0.

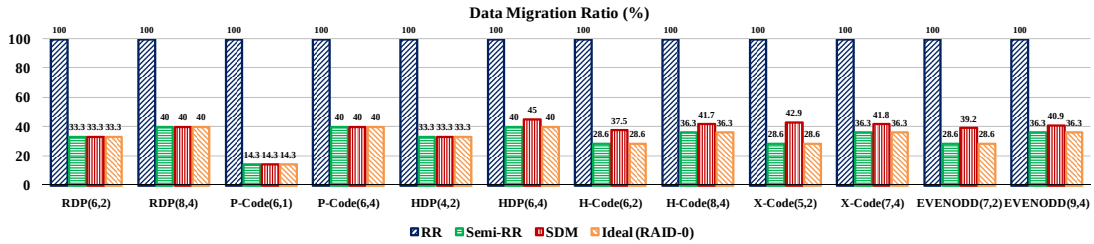


Figure 4.9: Comparison on data migration ratio under various RAID-6 scaling approaches.

Parity Modification Ratio

Third, parity modification ratio (R_p) among various RAID-6 scaling approaches under different cases is presented in Figure 4.10. Compared to other schemes with the same p and m, SDM sharply decreases the number of modified parities by up to 96.2%.

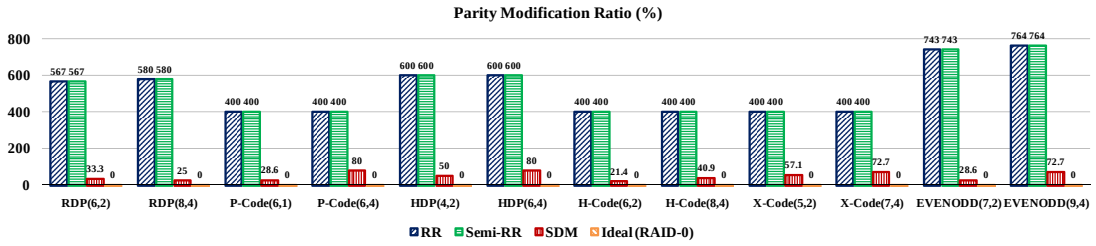


Figure 4.10: Comparison on parity modification ratio under various RAID-6 scaling approaches.

Computation Cost

We calculate the total number of XOR operations under various cases as shown in Figure 4.11. By using RR-based approaches, various codes have similar computation cost. SDM scheme decreases more than 80% computation cost compared to other approaches.

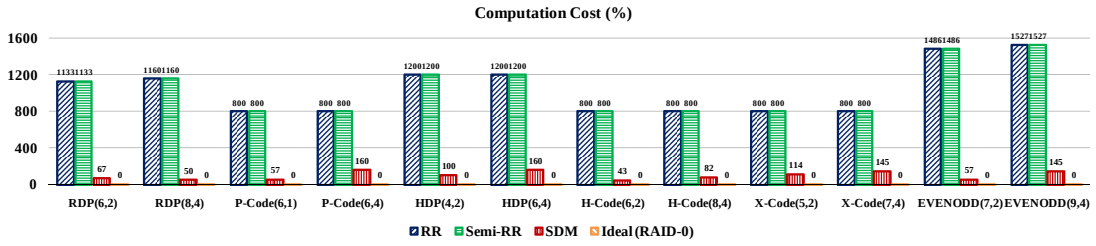


Figure 4.11: Comparison on computation cost under various RAID-6 scaling approaches (The number of B XORs is normalized to 100%).

Total Number of I/O Operations

Next, total number of I/O operations is calculated in these cases. If we use B as the baseline, the results of total I/Os are shown in Figure 4.12. By using SDM scheme, $72.7\% - 91.1\%$ I/Os are reduced.

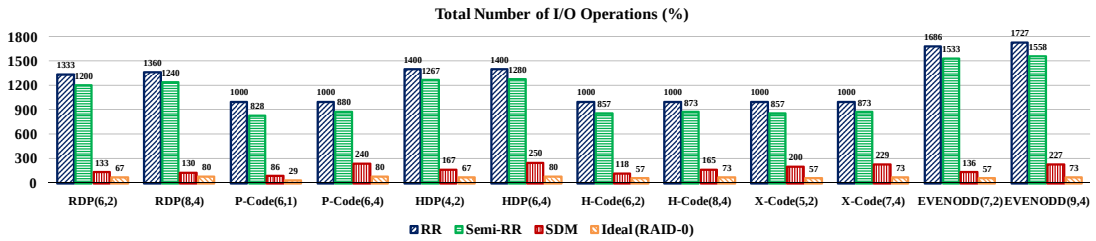


Figure 4.12: Comparison on total I/Os under various RAID-6 scaling approaches (The number of B I/O operations is normalized to 100%).

Migration Time

Migration time is evaluated as shown in Figure 4.13 and summarized in Table 4.8. Compared to other approaches, SDM performs well in multiple disks extension and decreases the migration time by up to 96.9%, which speeds up the scaling process to a factor of 32.

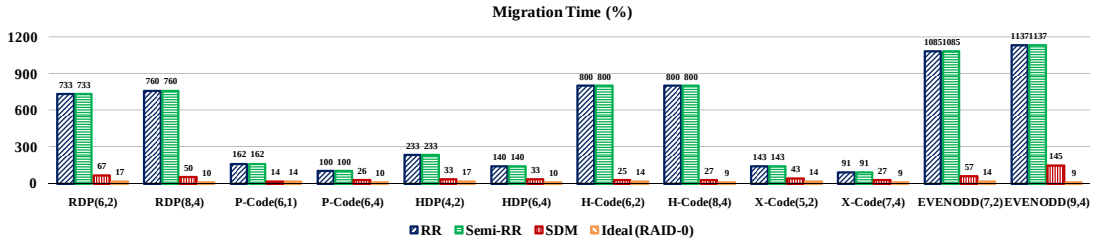


Figure 4.13: Comparison on migration time under various RAID-6 scaling approaches (The time $B * T_b$ is normalized to 100%).

Table 4.8: Speed Up of SDM Scheme over Other RAID-6 Scaling Schemes in terms of Migration Time

m & p	RDP	P-Code	HDP	H-Code	X-Code	EVENODD
$m = 2$ $p = 5$	$10.9\times$	–	$7.1\times$	$32.0\times$	$3.3\times$	$19.0\times$
$m = 4$ $p = 7$	$15.2\times$	$3.8\times$	$4.2\times$	$29.6\times$	$3.4\times$	$7.8\times$

4.4.3 Analysis

From the results in Section 4.4.2, compared to RR, Semi-RR and ALV, SDM has great advantages. There are several reasons to achieve these gains. First, SDM scheme is a global management on multiple stripes according to the priorities of data movements, which can minimize the parity modification cost, computation cost, total I/Os, etc. Second, compared to other approaches, SDM scheme distributes the migration I/O more evenly among data and parity disks, which can accelerate the scaling process in parallel. That is why SDM has better effects in horizontal codes which suffer unbalanced I/O. Third, although SDM sacrifices a small portion of stripes in each stripe set, it helps SDM to maintains a uniform workload, which creates favorable conditions for the storage system after scaling. SDM also has potential to have positive impacts on migration by aggregating small I/Os as ALV [89] and FastScale [90].

4.5 Summary

In this chapter, we have proposed a Stripe-based Data Migration (SDM) scheme to achieve high scalability for RAID-6. Our comprehensive mathematic analysis shows that SDM achieves better scalability compared to other approaches in the following aspects: 1) lower computation cost by reducing more than 80% XOR calculations; 2) less I/O operations by

72.7%-91.1%; and 3) shorter migration time and faster scaling process by a factor of up to 32.

Chapter 5

MDS Coding Scaling Framework (MDS-Frame)

5.1 Introduction

With the popularity of cloud computing [2], existing RAID-6 systems are difficult to meet these increasing requirements, where scalability plays a significant role. Fast scaling process also decreases the downtime cost of computer systems [56, 90]. Previous solutions [29, 55, 88] in RAID-6 are based on single MDS code, which has a large granularity on the extending number of disks. For example, it is impossible for scaling an existing disk array using RDP code from 6 to 7 disks. That's because RDP code only supports $p+1$ disks (e.g., 6 or 8 disks), where p is a prime number. Second, previous scaling approaches are focus on scale-up (adding disks), while scale-down (removing disks) is also important because removing inefficient disks can save energy consumption. Third, in a scaling process, most previous approaches [29, 55, 88, 28, 89, 37, 90] cannot estimate high overhead on disk

I/Os, such as parity modification. That's reasonable because these approaches are focus on RAID-0 or RAID-5, while RAID-6 has a more complex parity layout.

To address these challenges and to integrate techniques in previous chapters, in this chapter, we propose an **MDS** Code Scaling **Frame**work called **MDS-Frame**, which establishes the relationships and enables bidirectional scaling among various MDS codes. It consists of three layers: a management layer, an intermediate code layer and an MDS code repository.

The rest of this paper continues as follows: Section 5.2 discusses the motivation of our work. MDS-Frame is described in Section 5.3 to 5.5. Section 5.6 gives the quantitative analysis on scalability. Finally we conclude this chapter in Section 5.7.

5.2 Motivation

Besides the features discussed in Section 1.3.4, two new features need to be added to address the challenges in Section 5.1.

- Feature 5 (*Bidirectional Scaling*): A desired scaling approach should support both scale-up (adding disks) and scale-down (removing disks).
- Feature 6 (*Minimal Scaling Granularity*): The minimum number of adding/removing disk(s) could be one.

Based on SDM scheme (introduced in last chapter), it is possible to scaling from one code to another to avoid phenomena on the scalability problem, which motivates us to investigate the similarities among these codes and thus speed up the scaling process.

5.3 MDS-Frame

To overcome the scalability problem of existing MDS codes, we present an **MDS** Code Scaling **Frame**work (**MDS-Frame**), which is a unified management on MDS codes and provides flexible bidirectional scaling among these codes. We use "→" and "←" to delegate scale-up (adding disks) and scale-down (removing disks) between any two codes. For example, "RDP→EVENODD(6,7)" represents scaling from a 6-disk array using RDP to a 7-disk array using EVENODD. "RDP←EVENODD" is a scale-down process from EVENODD to RDP, and they have a same p value if disk number is not dedicated.

As shown in Figure 5.1, MDS-Frame has three layers, a management layer, an intermediate code layer and an MDS code repository.

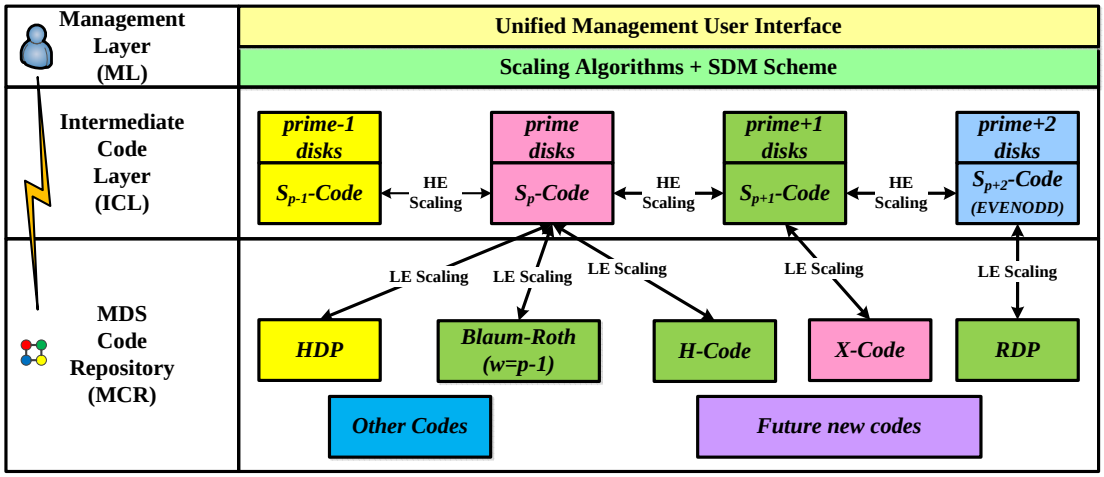


Figure 5.1: MDS-Frame Architecture.

Management Layer (ML) consists of a unified management user interface on MDS codes, scaling algorithms of these codes and SDM scheme. The unified management user interface provides the information of MDS codes and processes adding a new code or deleting an existing code in MDS-Frame. SDM scheme and scaling algorithms are also provided in this layer.

Intermediate Code Layer (ICL) includes four **S**calable Intermediate **Codes** (**S-Codes**), which are RAID-6 solutions for $p-1$, p, $p+1$ and $p+2$ disks. The corresponding codes are called S_{p-1}-Code, S_p-Code, S_{p+1}-Code and S_{p+2}-Code, respectively. Due to the similarities among S-Codes, a premier advantage of S-Codes is that high flexible bidirectional scaling can be achieved between any two codes by adding or deleting one disk. They act as a scaling highway to connect various MDS codes.

Existing MDS codes are stored in MDS Code Repository (MCR), such as EVENODD [8], RDP [18], H-Code, HDP code, etc. After a new MDS code is added into the MDS code repository, the links between the new code and intermediate codes will be established.

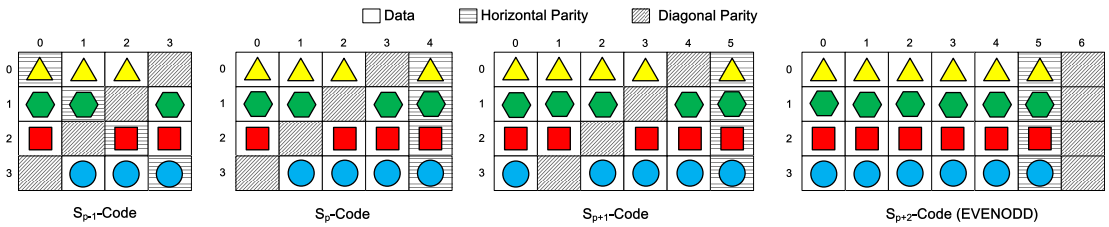
To clearly illustrate the efficiency of scaling process between/among various coding methods, we define two types of scaling (the quantitative definition will be given in Section 5.5.2), which are High Efficiency Scaling (HE) and Low Efficiency Scaling (LE). HE Scaling have low overhead on bidirectional scaling between two MDS codes, which includes data/parity migration, parity modification, computation cost, etc.

Detailed descriptions of ML and ICL are given in next two sections.

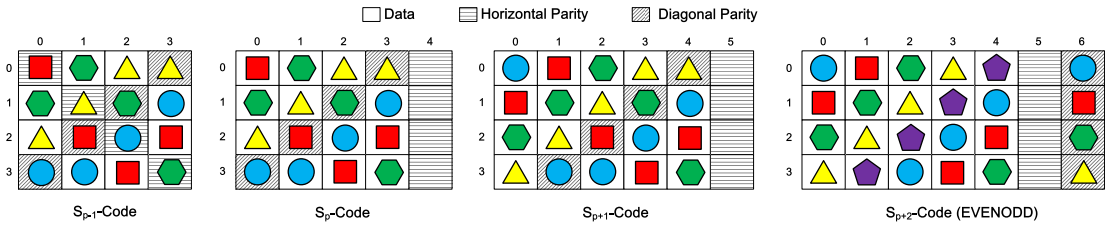
5.4 Intermediate Code Layer

In this section, S-Codes are introduced to improve the scalability, which act as intermediate codes in MDS-Frame. They are similar to each other and provide a comprehensive solution for $p-1$, p, $p+1$ and $p+2$ disks as follows (shown in Figure 5.2),

- **For $p-1$ disks**: S_{p-1}-Code, which is a variant of HDP code (Paper *III*, introduced in Chapter 3);
- **For p disks**: S_p-Code, which is a new code;



(a) Horizontal parity coding.



(b) Diagonal parity coding.

Figure 5.2: S-Codes ($p = 5$).

- **For $p + 1$ disks**: S_{p+1}-Code, which is a variant of H-Code(Paper *IV*, introduced in Chapter 2);

- **For $p + 2$ disks**: S_{p+2}-Code, which is the same as EVENODD code [8].

RDP, EVENODD, H-Code and HDP are presented in previous literatures [18] [8] and Chapters 2 and 3, respectively. So in this section, we briefly list the encoding/decoding equations of S_{p-1}-Code, S_p-Code and S_{p+1}-Code.

5.4.1 S_p-Code

Data/Parity Layout and Encoding

As shown in Figure 5.2, S_p-Code is represented by a $(p - 1)$-row-p-column matrix with a total of $(p - 1) * p$ elements. There are three types of elements in these matrices: ***data elements***, ***horizontal parity elements***, and ***diagonal parity elements***. The last column of S_p-Code is used for horizontal parity. Excluding the last column, the remaining matrices

are $(p-1)$-row-$(p-1)$-column square matrices. The diagonal part of these square matrices represents the diagonal parity of S_p-Code.

Assume $C_{i,j}$ $(0 \leq i \leq p-2, 0 \leq j \leq p-1)$ represents the element at the ith row and the jth column. Horizontal parity and diagonal parity elements of S_p-Code are constructed based on the following encoding equations,

Horizontal parity of S_p-Code:

$$C_{i,p-1} = \sum_{j=0}^{p-2} C_{i,j} \quad (j \neq p-2-i) \tag{5.1}$$

Diagonal parity of S_p-Code:

$$C_{i,p-2-i} = \sum_{j=0}^{p-2} C_{\langle p-3-i-j \rangle_p, j} \quad (j \neq p-2-i) \tag{5.2}$$

Figure 5.2 shows an example of S_p-Code for a 5-disk array ($p = 5$). It is a 4-row-5-column matrix. Column 4 is used for horizontal parity and the diagonal elements ($C_{0,3}$, $C_{1,2}$, $C_{2,1}$, etc.) are used for diagonal parity.

The horizontal parity encoding of S_p-Code is shown in Figure 5.2. We use different shapes to indicate different horizontal parity chains. Based on Equation 5.1, all horizontal parity elements could be encoded. For example, the horizontal parity element $C_{0,6}$ can be calculated by $C_{0,0} \oplus C_{0,1} \oplus C_{0,2}$. The element $C_{0,3}$ is not involved in this example because of $j = p-2-i$.

The diagonal parity encoding of S_p-Code is given in Figure 5.2. Different diagonal parity chains are also distinguished by various shapes. According to Equation 5.2, the diagonal parity elements can be got through modular arithmetic and XOR operations. For example, to calculate the diagonal parity element $C_{1,2}$ $(i = 1)$, first all the data elements

in the same diagonal parity chain should be selected ($C_{\langle p-3-i-j\rangle_p,j}$). If $j = 0$, based on Equation 5.2, $p-3-i-j = 1$ and $\langle 1\rangle_p = 1$, so the first data element is $C_{1,0}$. The following data elements which take part in XOR operations can be calculated similarly (the following data elements are $C_{0,1}$ and $C_{3,3}$). Second, the diagonal parity element ($C_{1,4}$) is constructed by performing an XOR operation on these data elements, i.e., $C_{1,2} = C_{1,0} \oplus C_{0,1} \oplus C_{3,3}$.

Construction Process

Based on the above data/parity layout and encoding scheme, the construction process of S_p-Code is straightforward.

- Label all data elements.
- Calculate both horizontal and diagonal parity elements according to the Equations 5.1 and 5.2.

Proof of Correctness

To prove that S_p-Code is correct, we consider one stripe in S_p-Code. The reconstruction of multiple stripes is just a matter of scale and similar to the reconstruction of one stripe. In a stripe, we have the following lemma and theorem,

Lemma 5.1. *We can find a sequence of a two-integer tuple (T_k, T'_k) where*

$$T_k = \left\langle p - 2 + \frac{k+1+\frac{1+(-1)^k}{2}}{2}(f_2 - f_1) \right\rangle_{p-1},$$
$$T'_k = \frac{1+(-1)^k}{2} f_1 + \frac{1+(-1)^{k+1}}{2} f_2 \quad (k = 0, 1, \cdots, 2p-3)$$

with $0 < f_2 - f_1 < p - 1$, all two-integer tuples $(0, f_1)$, $(0, f_2)$, $\cdots$, $(p-2, f_1)$, $(p-2, f_2)$ occur exactly once in the sequence.

Similar proof of this lemma can be found in many literatures in RAID-6 codes such as [8, 18, 86, 45].

Theorem 5.1. *A $(p-1)$-row-p-column stripe constructed according to the formal description of S_p-Code can be reconstructed under concurrent failures from any two columns.*

Proof. There are two cases, depending on dedicated horizontal parity column fails or not.

Case I: Double failures, one is from the horizontal parity column, the other is from a data column.

From the construction of S_p-Code, any two of the lost data elements cannot share a same diagonal parity chain. Therefore, any lost data elements can be recovered through the diagonal parity chains. After all lost data elements are recovered, the lost parity elements can be reconstructed using the above Equations 5.1 and 5.2.

Case II: Double failures of any two data columns.

We assume that the two failed columns are f_1 and f_2, where $0 < f_1 < f_2 < p-1$.

From the encoding of S_p-Code, any horizontal parity chain (in the ith row) includes all elements in the same row except $C_{i,p-2-i}$, which is served as diagonal parity. Thus two horizontal parity chains only contain one lost data element for each (C_{p-2-f_2,f_1} and C_{p-2-f_1,f_2}), which can be recovered by the retained elements.

For the failed columns f_1 and f_2, if an data element C_{i,f_2} on column f_2 could be reconstructed by its horizontal parity chain, then we can reconstruct the missing data element $C_{\langle i+f_2-f_1\rangle_p,f_1}$ who shares the same diagonal parity chain. Similarly, if a data element C_{i,f_1} on column f_1 could be recovered from its horizontal parity chain, the missing data element $C_{\langle i+f_2-f_1\rangle_p,f_2}$ who shares the same diagonal parity can be recovered. Then the reconstruction continues until all data elements are recovered. In this reconstruction

process, the reconstruction sequence is based on the sequence of the two-integer tuple shown in Lemma 3.1. Finally, the missing diagonal parity elements can be reconstructed according to Equation 5.2.

In conclusion, S_p-Code can be reconstructed under concurrent failures of any two columns. □

Reconstruction

We first consider how to recover a missing data element S_p-Code since any lost parity element can be recovered based on Equations 5.1 and 5.2. If a horizontal parity chain retains $p-1$ elements (including the parity element), the missing data element (assume it's C_{i,f_1} in column f_1 and $0 \leq f_1 \leq p-1$) in this chain can be reconstructed by the following equation,

$$C_{i,f_1} = \sum_{j=0}^{p-1} C_{i,j} \quad (j \neq p-2-i \quad and \quad j \neq f_1) \tag{5.3}$$

If a diagonal parity chain retains $p-2$ elements (including the parity element) and has a lost data element (C_{i,f_1}), the diagonal parity element (assume is in row r and represented by $C_{r,p-2-r}$ based on Equation 5.2) can be expressed as,

$$r = \langle p-3-i-f_1 \rangle_p \tag{5.4}$$

Based on Equation 5.2, the lost data element can be recovered by,

$$\begin{aligned} C_{i,f_1} &= C_{r,p-2-r} \oplus \sum_{j=0}^{p-2} C_{\langle i+f_1-j \rangle_p, j} \\ &(j \neq f_1 \; and \; j \neq \langle i+1+f_1 \rangle_p) \end{aligned} \tag{5.5}$$

Based on the Equations 5.1 to 5.5, we can easily recover all lost elements with any single disk failure. If two disks fail, as the proof of Theorem 2.1, there are two cases in our reconstruction process of S_p-Code based on horizontal parity column fails or not. A reconstruction example is given in Figure 5.3, which shows how to recover double columns failures by two recovery chains.

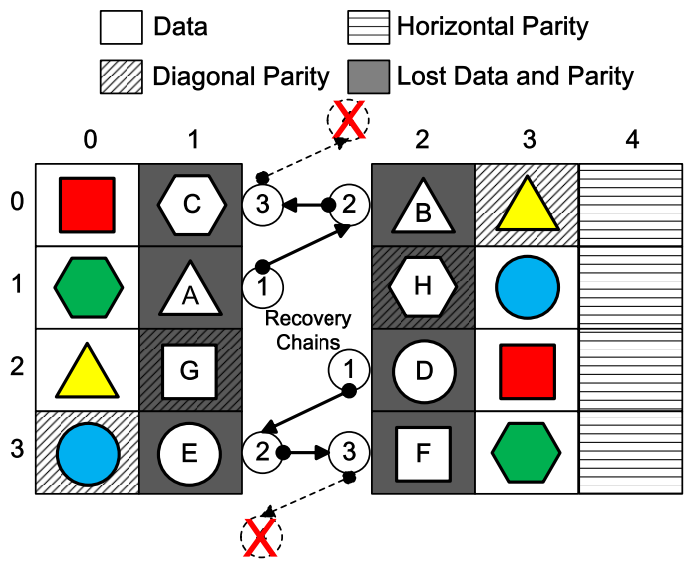


Figure 5.3: Reconstruction by two recovery chains in S_p-Code.

In this figure, there are double failures in columns 1 and 2: First we identify the two starting points of recovery chain: data elements A and D. Second we reconstruct data elements according to the corresponding recovery chains until they reach the endpoints (data elements C and F and the chains end by two "Xs" here). The orders to recover data elements are: one is A→B→C, the other is D→E→F. Finally we reconstruct diagonal parity elements G and H according to Equation 5.2.

5.4.2 S_{p-1}-Code and S_{p+1}-Code

Here we only list the equations of encoding/decoding of S_{p-1}-Code and S_{p+1}-Code, which are similar to S_p-Code.

Encoding equations of S_{p-1}-Code are,

$$\begin{cases} C_{i,i} = \sum_{j=0}^{p-2} C_{i,j} \quad (j \neq i \; and \; j \neq p-2-i) \\ C_{i,p-2-i} = \sum_{j=0}^{p-2} C_{\langle p-3-i-j \rangle_p, j} \quad (j \neq p-2-i) \end{cases} \tag{5.6}$$

Decoding equations of S_{p-1}-Code are (assume the lost data element is C_{i,f_1} and its corresponding diagonal parity element is $C_{r,p-2-r}$),

$$\begin{cases} C_{i,f_1} = \sum_{j=0}^{p-1} C_{i,j} \quad (j \neq f_1 \; and \; j \neq p-2-i) \\ r = \langle p-3-i-f_1 \rangle_p \\ C_{i,f_1} = C_{r,p-2-r} \oplus \sum_{j=0}^{p-2} C_{\langle i+f_1-j \rangle_p, j} \\ (j \neq f_1 \; and \; j \neq \langle i+f_1+1 \rangle_p) \end{cases} \tag{5.7}$$

Encoding equations of S_{p+1}-Code are,

$$\begin{cases} C_{i,p} = \sum_{j=0}^{p-1} C_{i,j} \quad (j \neq p-1-i) \\ C_{i,p-1-i} = \sum_{j=0}^{p-1} C_{\langle p-2-i-j \rangle_p, j} \quad (j \neq p-1-i) \end{cases} \tag{5.8}$$

Decoding equations of S_{p+1}-Code are (assume the lost data element is C_{i,f_1} and its corresponding diagonal parity element is $C_{r,p-1-r}$),

$$\begin{cases} C_{i,f_1} = \sum_{j=0}^{p} C_{i,j} \quad (j \neq f_1 \; and \; j \neq p-1-i) \\ r = \langle p-2-i-f_1 \rangle_p \\ C_{i,f_1} = C_{r,p-1-r} \oplus \sum_{j=0}^{p-1} C_{\langle i+f_1-j \rangle_p, j} \\ (j \neq f_1 \; and \; j \neq \langle i+f_1+1 \rangle_p) \end{cases} \tag{5.9}$$

5.4.3 Observations

As shown in Figure 5.2, S-Codes are represented by a $(p-1)$-row-$(p+x-2)$-column matrix with a total of $(p-1)*(p+x-2)$ elements (x=-1,0,1,2). There are three types of elements in these matrices: ***data elements***, ***horizontal parity elements***, and ***diagonal parity elements***. By investigating the similarities among S-Codes, we have the following observations,

Observation 5.1. *(variation of number of elements.) If an intermediate code in MDS-Frame supports $(p+x)$ disks (x=-1,0,1,2), the number of data elements per stripe is $(p-1)*(p+x-2)$, the number of parity elements per stripe is $2*(p-1)$.*

Observation 5.2. *(variation of parity chain length and parity disks) If an intermediate code in MDS-Frame supports $(p+x)$ disks (x=-1,0,1,2), the horizontal parity chain length is equal to $p+x-1$ or $p+x$. With the increasing value of x, the number of parity chain length and dedicated parity disks will be increased.*

Observation 5.3. *(variation of parity encoding complexity.) If an intermediate code in MDS-Frame supports $(p+x)$ disks (x=-1,0,1,2), with the increasing value of $|x|$, the parity encoding complexity will be increased.*

Observation 5.4. *(variation of encoding/decoding performance) For intermediate codes in MDS-Frame supporting $(p+x)$ disks (x=-1,0,1,2), when p is large enough, they have the optimal or near-optimal encoding/decoding performance.*

5.5 Management Layer

In this section, Management Layer (ML) is introduced in detail. Because SDM scheme is presented in last chapter, in this section we mainly discuss the unified management user interface and scaling algorithms of these codes.

5.5.1 Unified Management User Interface

The unified management user interface provides information on MDS codes in MDS-Frame and processes adding or deleting codes. The information of MDS codes involves code name, the number of disks provided, encoding and decoding equations, etc. Adding a new code into MDS-Frame is shown in Algorithm 5.1.

Algorithm 5.1: Add A New Code into MDS-Frame

Step 1: Get the information of the new code (assume it is called "Code-X1", which supports $p + x_1$ disks).
Step 2: Establish the relationship between Code-X1 and intermediate codes (S-Codes).
if $-1 \leq x_1 \leq 2$ **then**
  Step 2A: Compare the layouts between Code-X1 and S_{p+x}-Code ($x = x_1 - 1, x_1 + 1$ where $-1 \leq x \leq 2$) sequentially and calculate the cost of scaling process.
  Step 2B: According to the cost, establish a link and label the scaling link (HE or LE).
end

5.5.2 Definition of the HE and LE Scaling

Suppose the total number of data elements in a disk array is B. According to the results in RAID-0 scaling [90], by adding m disks to a disk array with n disks, the optimal data migration under uniform data distribution is $\frac{mB}{n+m}$. Similarly, deleting m disk from a disk

array with n disks, the optimal data migration is $\frac{|m|B}{n}$. In RAID-6, the capacity of two disks are used for parities and the actual data disks is $n-2$, so the optimal data and parity migration for adding/deleting m disks with uniform distribution is $\frac{mB}{n-2+m}$, $\frac{|m|B}{n-2}$, respectively. Thus the average amount of data and parity migration by scaling one disk (scale-up and scale-down) is,

$$\frac{1}{2}(\frac{B}{n-2+1}+\frac{B}{n-2}) \approx \frac{B}{n-2} \tag{5.10}$$

Based on Equation 5.10, we define HE scaling in terms of the amount of data and parity migration,

Definition 5.1. *For a bidirectional scaling between two random codes Code-X1 and Code-X2 supporting $p+x_1$ and $p+x_2$ disks ($-1 \leq x_1 \leq x_2 \leq 2$ and $|x_1-x_2| \leq 1$), if the amount of data and parity migration in any one-way scaling (Code-X1→Code-X2 and Code-X1←Code-X2) is no more than $B/(p+x_1-2)$, these scaling are HE scaling. Otherwise they are LE scaling.*

5.5.3 Scaling Algorithms

Scaling algorithms in MDS-Frame are presented by comparing the layouts of various codes. The lower overhead of the scaling is, the simpler scaling algorithm will be. In the following we show an example when a scaling from S_p-Code to S_{p+1}-Code with a same value of p (S_p-Code→S_{p+1}-Code). By comparing the layouts as shown in Figure 5.2, an efficient scaling algorithm is shown in Algorithm 5.2 (add an empty disk as the first column).

Algorithm 5.2: Scaling Algorithm (S_{p-1}-Code→S_p-Code)

Step 1: Get the information of the S_{p-1}-Code and S_p-Code.
Step 2: Label a new disk as column 4.
Step 3: Assume $C_{i,j}$ is a random element in a stripe based on the layout of S_{p-1}-Code and the corresponding element after scaling is $C_{i',j'}$ (the same stripe ID in S_p-Code).

```
if 0 ≤ i, j ≤ p − 2 then
    if i==j (horizontal parity elements) then
        move this element to column 4;
        i' = i, j' = 4.
    end
    else
        other elements are retained at the original position;
        i' = i, j' = j.
    end
end
```

Table 5.1: Overhead of Typical scaling in MDS-Frame

Transformations	**Migrated Data/Parities**	**Modified Parities**	**Total I/Os**
S_{p-1}-Code→S_p-Code	$B/(p-3)$	0	$2B/(p-3)$
S_{p-1}-Code←S_p-Code	0	$4B/(p-2)$	$10B/(p-2)$
S_p-Code→S_{p+1}-Code	0	0	0
S_p-Code←S_{p+1}-Code	$B/(p-1)$	$4B/(p-1)$	$10B/(p-1)$
S_{p+1}-Code→S_{p+2}-Code	$B/(p-1)$	$B/(p-1)$	$4B/(p-1)$
S_{p+1}-Code←S_{p+2}-Code	0	$4B/p$	$10B/p$

5.5.4 Overhead Analysis of Scaling between Intermediate Codes

We use three metrics to measure the overhead of data/parity migration, parity modification and total I/Os: 1) total number of migrated data and parities; 2) total number of modified parities; 3) total number of I/O operations. And we calculate the scaling between adjunct intermediate codes with a same value of p. The results are shown in Table 5.1 and show that scaling satisfies the condition of high efficiency (HE).

5.5.5 Scaling Rules

For a random MDS codes (assume it is called "Code-X", which supports $p + x$ disks) in MDS-Frame, we have the following rules by scaling m disks,

1) if $-1 \leq x + m \leq 2$, scale m times by adding or removing one disk once. The scaling targets are intermediate codes, which provide HE scaling. The scaling sequence is Code-X→intermediate Code→intermediate Code→...→intermediate Code.

2) if $x + m < -1$ or $x + m > 2$, we need to find a proper p' which satisfies $-1 \leq x + m - (p' - p) \leq 2$. First we scale from $p + x$ disks to $p' + x$ disks by using Code-X, existing fast scaling approaches [29, 55, 88] can be used to accelerate the process. Second repeat the scaling process as the rule 1. The scaling sequence is Code-X ($p + x$ disks)→Code-X ($p' + x$ disks)→intermediate Code→intermediate Code→...→intermediate Code.

For example, if we want to scale from a 6-disk array using HDP code to a RAID-6 array with 9 disks, we can follow the scaling sequence: HDP→S_p-Code→S_{p+1}-Code→S_{p+2}-Code(EVENODD).

5.6 Scalability Analysis

In this section, we evaluate scaling cost between various codes in MDS-Frame to demonstrate its effectiveness on scalability.

5.6.1 Evaluation Methodology

We compare the scaling in MDS-Frame to Round-Robin (RR) [29, 55, 88], which provides RAID-6 bidirectional scaling. Other approaches [28, 89, 37, 90] don't support bidirectional scaling in RAID-6.

In our comparison, Round-Robin (RR) approach is optimized to adopt the layout of MDS codes in RAID-6. All data elements are migrated in round-robin order, and the parities are updated when a data element is moved into (or out from) the corresponding parity chains. Different from MDS-Frame, all MDS codes in RR approach are scaled directly without using any immediate codes.

In addition to the metric of total I/Os (introduced in Section 5.5.4), we also use the following two metrics,

1) total number of XOR operations: It is used to measure the computation cost in the scaling process;

2) migration time: If we ignore the remapping time and computation time, then assume the same time on a read or write request to a data/parity element (denoted by T_e). Suppose the migration I/O can be processed in parallel on each disk. We can calculate the migration time for each scaling process which reflects the effectiveness of the scalability.

5.6.2 Numerical Results and Analysis

Here we give the numerical results of MDS-Frame compared to RR using above metrics.

1) Total number of I/O operations: First, the total number of I/Os for different scaling are shown in Figure 5.4. We notice that MDS-Frame reduces more than 44.1% and up to 97.4% I/O cost compared to RR.

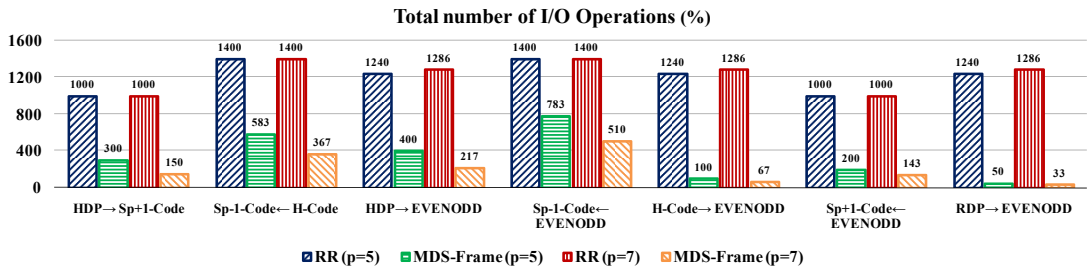


Figure 5.4: Total I/Os of various scaling processes in MDS-Frame vs. Round-Robin when $p = 5$ and $p = 7$ (The number of B I/O operations is normalized to 100%).

2) Total number of XOR operations: Second, we calculate the total number of XOR operations under various cases as shown in Figure 5.5, where MDS-Frames reduces up to 98.5% computation cost compared to RR.

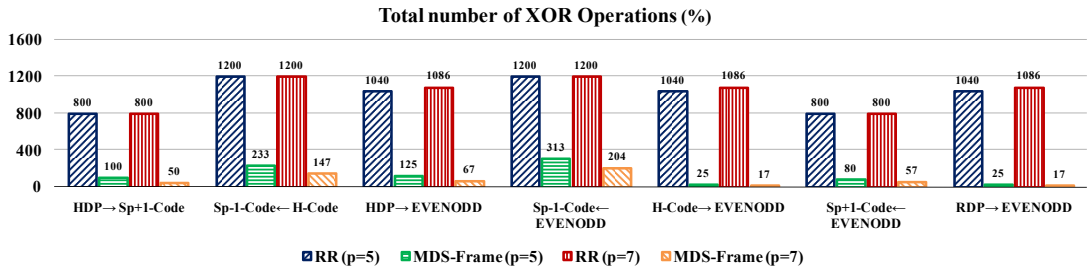


Figure 5.5: Computation cost of various scaling processes in MDS-Frame vs. Round-Robin when $p = 5$ and $p = 7$ (The number of B XORs is normalized to 100%).

3) Migration time: Next, Migration time is evaluated as shown in Figure 5.6. Compared to Round-Robin, SDM-Frame performs better with lower the migration time by up to 95.2%, which can speed up the scaling process to a factor of 20.7.

4) Value of p: Finally, we select a scaling HDP→EVENODD and evaluate the migration time under different value of p as shown in Figure 5.7. It is clear that MDS-Frame keeps a gradual downward trend on overhead with the increasing value of p.

Above results show that compared to RR approach, MDS-Frame presents great advantage on scalability among various MDS codes. There are several reasons to achieve

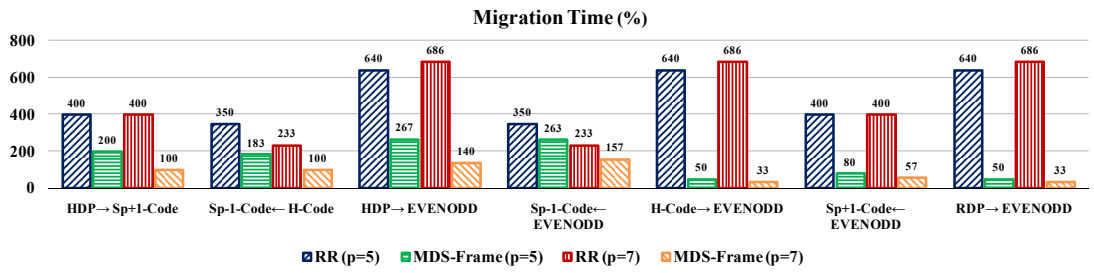


Figure 5.6: Migration time of various scaling processes in MDS-Frame vs. Round-Robin when $p = 5$ and $p = 7$ (The time $B * T_e$ is normalized to 100%).

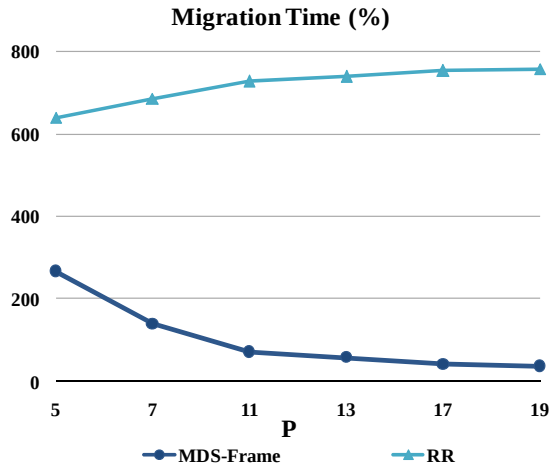


Figure 5.7: Migration time in scaling HDP→EVENODD under various p (the number of B I/O operations is normalized to 100%).

these gains. First, MDS-Frame is a unified management for all MDS codes, which select the optimal scaling path with the lowest cost. Second, the intermediate code provides a scaling highway with low overhead, which connects various horizontal codes. Third, the scaling algorithms in MDS-Frame is based on the minor differences of the layouts among various codes, which guarantees high efficiency scaling processes. All these aspects can minimize the movement of data and parities thus reduces total overhead.

5.7 Summary

In this chapter, we propose an MDS Code Scaling Framework (MDS-Frame) to improve the scalability of RAID-6. Our comprehensive mathematic analysis shows that MDS-Frame achieves high scalability in the following aspects: 1) reduced I/Os by up to 97.4% due to less migration data and modified parities; 2) lowered computation cost in terms of less XOR calculations by up to 98.5%; 3) shorter migration time by up to 95.2%, and 4) faster scaling process by a factor of up to 20.7.

Chapter 6

Conclusions

In this dissertation, we make the following contributions to improve the performance and scalability of RAID-6 systems using erasure codes:

1. We propose a novel and efficient XOR-based RAID-6 code (H-Code) to offer optimal partial stripe write performance among all MDS codes, which is demonstrated by analyzing the overhead of partial stripe write to multiple data elements with a quantitative approach. We prove that H-Code has not only the optimal property demonstrated by vertical MDS codes including storage efficiency, encoding/decoding computational complexity and single write complexity, but also the optimized partial stripe write complexity to multiple data elements.

2. We propose a novel and efficient XOR-based RAID-6 code (HDP Code) to offer not only the property provided by typical MDS codes such as optimal storage efficiency, but also best load balancing and high reliability due to horizontal-diagonal parity.

3. We propose a new data migration scheme (SDM) to address RAID-6 scalability problem, a significant issue in large scale data storage systems. SDM accelerates

RAID-6 scaling process, in terms of the number of modified parities, the number of XOR calculations, the total number of I/O operations and the migration time. SDM balances I/O distribution among multiple disks in a disk array, reducing the migration time indirectly. And SDM provides fast data addressing algorithms.

4. We propose a novel MDS code scaling framework (MDS-Frame), which is a unified management scheme on MDS codes to allow flexible scaling from one code to another.

5. We quantitatively analyze the cost of performance and scalability among various MDS codes, and prove that H-Code, HDP, SDM and MDS-Frame are cost-effective solutions to improve the performance and scalability of RAID-6 systems.

6. We summarize the pros and cons of several MDS codes and give some insightful observations, which can help guide future code design.

Bibliography

[1] ALLMYDATA. Unlimited online backup, storage, and sharing. http://allmydata.com, 2008. 1

[2] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, UC Berkeley, February 2009. 2, 70, 95

[3] B. Nisbet. FAS storage systems: Laying the foundation for application availability. http://www.netapp.com/us/library/analyst-reports/ar1056.html, February 2008. 1

[4] B. Welch and M. Unangst and Z. Abbasi and G. Gibson and B. Mueller and J. Small and J. Zelenka and B. Zhou. Scalable Performance of the Panasas Parallel File System. http://www.panasas.com/sites/default/files/uploads/docs/panasas_scalable_storage_arch_wp_1066.pdf, February 2008. 1

[5] E. Bachmat and Y. Ofek. Load balancing method for exchanging data in different physical disk storage devices in a disk array stroage device independently of data

processing system operation. US Patent No. 6237063B1, May 2001. 2, 10, 48

[6] E. Bachmat, Y. Ofek, A. Zakai, M. Schreiber, V. Dubrovsky, T. Lam, and R. Michel. Load balancing on disk array storage device. US Patent No. 6711649B1, March 2004. 2, 10, 48

[7] M. Beck, D. Arnold, A. Bassi, F. Berman, H. Casanova, J. Dongarra, T. Moore, G. Obertelli, J. Plank, M. Swany, S. Vadhiyar, and R. Wolski. Logistical computing and internetworking: Middleware for the use of storage in communication. In *Proc. of the 3rd Annual International Workshop on Active Middleware Services (AMS)*, San Francisco, CA, August 2001. 2

[8] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on Computers*, 44(2):192–202, February 1995. 2, 5, 6, 19, 26, 37, 51, 56, 60, 86, 98, 99, 102

[9] M. Blaum, J. Hafner, and S. Hetzler. Partial-MDS codes and their application to RAID type of architectures. *CoRR*, abs/1205.0997, 2012. 1, 5

[10] M. Blaum and R. Roth. On lowest density MDS codes. *IEEE Transactions on Information Theory*, 45(1):46–59, January 1999. 2, 5, 6, 19, 20

[11] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An XOR-based Erasure-Resilient coding scheme. Technical Report TR-95-048, International Computer Science Institute, August 1995. 2, 5, 19

[12] V. Bohossian, C. Fan, P. LeMahieu, M. Riedel, J. Bruck, and L. Xu. Computing in the RAIN: A reliable array of independent nodes. *IEEE Transactions on Parallel and Distributed Systems*, 12(2):99–114, February 2001. 2

[13] J. Bonwick. RAID-Z. http://blogs.sun.com/bonwick/entry/raidz, 2010. 2, 9, 16, 19

[14] A. Brinkmann, K. Salzwedel, and C. Scheideler. Efficient, distributed data placement strategies for storage area networks. In *Proc. of the ACM SPAA'00*, Bar Harbor, ME, July 2000. 16

[15] Y. Cassuto and J. Bruck. Cyclic lowest density MDS array codes. *IEEE Transactions on Information Theory*, 55(4):1721–1729, April 2009. 2, 5, 8, 19, 20, 22, 32, 33, 34, 36, 47, 59, 68

[16] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994. 1

[17] CLEVERSAFE, Inc. Cleversafe dispersed storage. Open source code distribution: http://www.cleversafe.org/downloads, 2008. 1

[18] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-Diagonal Parity for double disk failure correction. In *Proc. of the USENIX FAST'04*, San Francisco, CA, March 2004. 1, 2, 5, 7, 19, 26, 37, 47, 51, 56, 60, 68, 72, 74, 86, 98, 99, 102

[19] E. David. Method for improving partial stripe write performance in disk array subsystems. US Patent No. 5333305, July 1994. 2, 9, 19

[20] EMC Corporation. EMC CLARiiON RAID 6 Technology: A Detailed Review. http://www.emc.com/collateral/hardware/white-papers/h2891-clariion-raid-6.pdf, July 2007. 10, 60

[21] G. Feng, R. Deng, F. Bao, and J. Shen. New efficient MDS array codes for RAID part I: Reed-Solomon-like codes for tolerating three disk failures. *IEEE Transactions on Computers*, 54(9):1071–1080, September 2005. 5

[22] G. Feng, R. Deng, F. Bao, and J. Shen. New efficient MDS array codes for RAID part II: Rabin-like codes for tolerating multiple (≥ 4) disk failures. *IEEE Transactions on Computers*, 54(12):1473–1483, December 2005. 5

[23] C. Franklin and J. Wong. Expansion of RAID subsystems using spare space with immediate access to new space. US Patent 2003/0115412 A1, June 2003. 16

[24] H. Fujita and K. Sakaniwa. Modified Low-Density MDS array codes for tolerating double disk failures in disk arrays. *IEEE Transactions on Computers*, 56(4):563–566, April 2007. 2, 19

[25] R. Gallager. Low-density parity-check codes. *IRE Transactions on Information Theory*, 8(1):21–28, 1962. 5

[26] G. Ganger, B. Worthington, R. Hou, and Y. Patt. Disk subsystem load balancing: disk striping vs. conventional data placement. In *Proc. of the HICSS'93*, Wailea, HI, January 1993. 2, 10, 48

[27] S. Ghandeharizadeh and D. Kim. On-line reorganization of data in scalable continuous media servers. In *Proc. of the DEXA'96*, Zurich, Switzerland, September 1996. 70

[28] A. Goel, C. Shahabi, S. Yao, and R. Zimmermann. SCADDAR: An efficient randomized technique to reorganize continuous media blocks. In *Proc. of the ICDE'02*, San Jose, CA, February 2002. 2, 13, 14, 71, 86, 89, 95, 111

[29] J. Gonzalez and T. Cortes. Increasing the capacity of RAID5 by online gradual assimilation. In *Proc. of the SNAPI'04*, Antibes Juan-les-pins, France, September 2004. 2, 13, 71, 73, 86, 95, 110, 111

[30] K. Gopinath, N. Muppalaneni, N. Kumar, and P. Risbood. A 3-tier RAID storage system with RAID1, RAID5 and compressed RAID5 for Linux. In *Proc. of the USENIX ATC'00*, San Diego, CA, June 2000. 2, 9, 19

[31] K. Greenan, X. Li, and J. Wylie. Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. In *Proc. of the IEEE MSST'10*, Incline Village, NV, May 2010. 1, 5

[32] H. Anvin. The mathematics of RAID-6. `http://kernel.org/pub/linux/kernel/people/hpa/raid6.pdf`, May 2009. 5

[33] J. Hafner. WEAVER codes: Highly fault tolerant erasure codes for storage systems. In *Proc. of the USENIX FAST'05*, San Francisco, CA, December 2005. 1, 5

[34] J. Hafner. HoVer erasure codes for disk arrays. In *Proc. of the IEEE/IFIP DSN'06*, Philadelphia, PA, June 2006. 1, 3, 4, 5, 8

[35] J. Hafner, V. Deenadhayalan, K. Rao, and J. Tomlin. Matrix methods for lost data reconstruction in erasure codes. In *Proc. of the FAST'05*, San Francisco, CA, December 2005. 1

[36] E. Hashemi. Load balancing configuration for storage arrays employing mirroring and striping. US Patent No. 6425052B1, July 2002. 2, 10, 48

[37] S. Hetzler. Storage array scaling method and system with minimal data movement. US Patent 20080276057, June 2008. 2, 13, 14, 71, 86, 95, 111

[38] C. Huang, M. Chen, and J. Li. Pyramid Codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *Proc. of the IEEE NCA'07*, Cambridge, MA, July 2007. 1, 5

[39] C. Huang, J. Li, and M. Chen. On optimizing xor-based codes for fault-tolerant storage applications. In *Proc. of the IEEE Information Theory Workshop (ITW)*, Lake Tahoe, CA, September 2007. 1

[40] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in windows azure storage. In *Proc. of the USENIX ATC'12*, Boston, MA, June 2012. 1, 5

[41] C. Huang and L. Xu. STAR: An efficient coding scheme for correcting triple storage node failures. In *Proc. of the USENIX FAST'05*, San Francisco, CA, December 2005. 1, 5

[42] C. Huang and L. Xu. STAR: An efficient coding scheme for correcting triple storage node failures. *IEEE Transactions on Computers*, 57(7):889–901, July 2008. 1, 5

[43] K. Hwang, H. Jin, and R. Ho. RAID-x: A new distributed disk array for I/O-Centric cluster computing. In *Proc. of the HPDC'00*, Pittsburgh, PA, August 2000. 16, 70

[44] R. Jantz. Method for host-based I/O workload balancing on redundant array controllers. US Patent No. 5937428, August 1999. 2, 10, 48

[45] C. Jin, H. Jiang, D. Feng, and L. Tian. P-Code: A new RAID-6 code with optimal properties. In *Proc. of the ACM ICS'09*, Yorktown Heights, NY, June 2009. 2, 5, 8, 19, 20, 22, 26, 32, 33, 34, 36, 47, 48, 52, 56, 59, 60, 68, 72, 74, 86, 102

[46] H. Jin, X. Zhou, D. Feng, and J. Zhang. Improving partial stripe write performance in RAID level 5. In *Proc. of the 2nd IEEE International Caracas Conference on Devices, Circuits and Systems*, Isla de Margarita, Venezuela, March 1998. 2, 9, 19

[47] N. Joukov, A. Krishnakumar, C. Patti, A. Rai, S. Satnur, A. Traeger, and E. Zadok. RAIF: Redundant array of independent filesystems. In *Proc. of the IEEE MSST'07*, San Diego, CA, September 2007. 2

[48] D. Kenchammana-Hosekote, D. He, and J. Hafner. REO: A generic raid engine and optimizer. In *Proc. of the USENIX FAST'07*, San Jose, CA, February 2007. 1

[49] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang. Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads. In *Proc. of the USENIX FAST'12*, San Jose, CA, February 2012. 1, 5

[50] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. In *Proc. of the ASPLOS'00*, Cambridge, MA, November 2000. 2

[51] A. Kuratti. *Analytical Evaluation of the RAID 5 Disk Array*. PhD thesis, University of Arizona, 1994. 2, 9, 19

[52] M. Li, J. Shu, and W. Zheng. GRID codes: Strip-based erasure code with high fault tolerance for storage systems. *ACM Transactions on Storage*, 4(4):Article 15, January 2009. 5

[53] M. Luby, M. Mitzenmacher, M. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *Proc. of the ACM STOC'97*, El Paso, TX, May 1997. 5

[54] J. Luo, C. Huang, and L. Xu. Decoding star code for tolerating simultaneous disk failure and silent errors. In *Proc. of the IEEE/IFIP DSN'10*, Chicago, IL, June 2010. 1, 5

[55] N. Brown. Online RAID-5 Resizing. drivers/md/raid5.c in the source code of Linux Kernel 2.6.18. http://www.kernel.org/, September 2006. 2, 13, 14, 71, 73, 86, 95, 110, 111

[56] D. Patterson. A simple way to estimate the cost of down-time. In *Proc. of the LISA'02*, Philadelphia, PA, October 2002. 70, 95

[57] D. Patterson, G. Gibson, and R. Katz. A case for Redundant Arrays of Inexpensive Disks (RAID). In *Proc. of the SIGMOD'88*, Chicago, IL, June 1988. 1

[58] E. Pinheiro, W. Weber, and L. Barroso. Failure trends in a large disk drive population. In *Proc. of the USENIX FAST'07*, San Jose, CA, February 2007. 1

[59] J. Plank. Erasure codes for storage applications. In *Tutorial Slides presented at the USENIX FAST'05*, San Francisco, CA, December 2005. 2

[60] J. Plank. A new minimum density RAID-6 code with a word size of eight. In *Proc. of the IEEE NCA'08*, Cambridge, MA, July 2008. 5, 6

[61] J. Plank. The RAID-6 liberation codes. In *Proc. of the USENIX FAST'08*, San Jose, CA, February 2008. 2, 5, 6, 19

[62] J. Plank, A. Buchsbaum, and B. Zanden. Minimum density RAID-6 codes. *ACM Transactions on Storage*, 6(4):Article 16, May 2011. 6, 76

[63] J. Plank, J. Luo, C. Schuman, L. Xu, and Z. Wilcox-OHearn. A performance evaluation and examination of open-source erasure coding libraries for storage. In *Proc. of the USENIX FAST'09*, San Francisco, CA, February 2009. 1

[64] J. Plank and M. Thomason. A practical analysis of low-density parity-check erasure codes for wide-area storage applications. In *Proc. of the IEEE/IFIP DSN'04*, Florence, Italy, June 2004. 5

[65] J. Plank and L. Xu. Optimizing Cauchy Reed-Solomon codes for Fault-Tolerant network storage applications. In *Proc. of the IEEE NCA'06*, Cambridge, MA, July 2006. 6

[66] RANDOM.ORG. Random Integer Generator. http://www.random.org/integers/, 2010. 37, 62

[67] I. Reed and G.Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, pages 300–304, 1960. 2, 5, 19

[68] J. Resch and J. Plank. AONT-RS: Blending security and performance in dispersed storage systems. In *Proc. of the USENIX FAST'11*, San Jose, CA, February 2011. 1

[69] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5):40–49, September & October 2001. 2

[70] Y. Saito, S. Frolund, A. Veitch, A. Merchant, and S. Spence. FAB: Building distributed enterprise disk arrays from commodity components. In *Proc. of the ASPLOS'04*, Boston, MA, October 2004. 16, 70

[71] P. Scheuermann, G. Weikum, and P. Zabback. Data partitioning and load balancing in parallel disk systems. *the VLDB Journal*, 7(1):48–66, February 1998. 2, 10, 48

[72] B. Schroeder and G. Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proc. of the USENIX FAST'07*, San Jose, CA, February 2007. 1

[73] Storage Networking Industry Association (SNIA). RAID 6/RAID level 6. http://www.snia.org.cn/dic.php?word=r, 2010. 1

[74] M. Storer, K. Greenan, E. Miller, and K. Voruganti. Pergamum: Replacing tape with energy efcient, reliable, disk-based archival storage. In *Proc. of the USENIX FAST'08*, San Jose, CA, February 2008. 2

[75] A. Thomasian and J. Xu. Cost analysis of the X-code double parity array. In *Proc. of the IEEE MSST'07*, San Diego, CA, September 2007. 7

[76] L. Tian, D. Feng, H. Jiang, L. Zeng, J. Chen, Z. Wang, and Z. Song. PRO: A popularity-based multi-threaded reconstruction optimization for RAID-structured storage systems. In *Proc. of the USENIX FAST'07*, San Jose, CA, February 2007. 10

[77] S. Wan, Q. Cao, C. Xie, B. Eckart, and X. He. Code-M: A Non-MDS erasure code scheme to support fast recovery from up to two-disk failures in storage systems. In *Proc. of the IEEE/IFIP DSN'10*, Chicago, IL, June 2010. 3, 5, 9

[78] Shenggang Wan, Qiang Cao, Jianzhong Huang, Siyi Li, Xin Li, Shenghui Zhan, Li Yu, and Changsheng Xie. Victim disk first: An asymmetric cache to boost the performance of disk arrays under faulty conditions. In *Proc. of the USENIX ATC'11*, Portland, OR, June 2011. 11

[79] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou. Scalable performance of the Panasas parallel file system. In *Proc. of the USENIX FAST'08*, San Jose, CA, February 2008. 1

[80] Hadoop Wiki. HDFS RAID. http://wiki.apache.org/hadoop/hdfs-raid, 2011. 16

[81] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems*, 14(1):108–136, February 1996. 16

[82] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao. WorkOut: I/O workload outsourcing for boosting RAID reconstruction performance. In *Proc. of the USENIX FAST'09*, San Francisco, CA, February 2009. 10

[83] J. Wylie and R. Swaminathan. Determining fault tolerance of XOR-based erasure codes efficiently. In *Proc. of the IEEE/IFIP DSN'07*, Edinburgh, UK, June 2007. 1, 5

[84] L. Xiang, Y. Xu, J. Lui, and Q. Chang. Optimal recovery of single disk failure in RDP code storage systems. In *Proc. of the ACM SIGMETRICS'10*, New York, NY, June 2010. 10, 11, 51

[85] L. Xu, V. Bohossian, J. Bruck, and D. Wagner. Low-density MDS codes and factors of complete graphs. *IEEE Transactions on Information Theory*, 45(6):1817–1826, September 1999. 2, 5, 52

[86] L. Xu and J. Bruck. X-Code: MDS array codes with optimal encoding. *IEEE Transactions on Information Theory*, 45(1):272–276, January 1999. 2, 5, 7, 19, 20, 26, 32, 33, 34, 36, 47, 48, 51, 56, 59, 86, 102

[87] X. Yu, B. Gum, Y. Chen, R. Wang, K. Li, A. Krishnamurthy, and T. Anderson. Trading capacity for performance in a disk array. In *Proc. of the OSDI'00*, San Diego, CA, October 2000. 70

[88] G. Zhang, J. Shu, W. Xue, and W. Zheng. SLAS: An efficient approach to scaling round-robin striped volumes. *ACM Transactions on Storage*, 3(1):1–39, March 2007. 2, 13, 71, 73, 86, 95, 110, 111

[89] G. Zhang, W. Zheng, and J. Shu. ALV: A new data redistribution approach to RAID-5 scaling. *IEEE Transactions on Computers*, 59(3):345–357, March 2010. 2, 13, 14, 71, 86, 93, 95, 111

[90] W. Zheng and G. Zhang. FastScale: Accelerate RAID scaling by minimizing data migration. In *Proc. of the FAST'11*, San Jose, CA, February 2011. 2, 12, 13, 14, 71, 86, 89, 93, 95, 107, 111

[91] B. Zhu, K. Li, and H. Patterson. Avoiding the disk bottleneck in the data domain deduplication file system. In *Proc. of the USENIX FAST'08*, San Jose, CA, February 2008. 1

[92] Y. Zhu, P. Lee, L. Xiang, Y. Xu, and L. Gao. A cost-based heterogeneous recovery scheme for distributed storage systems with RAID-6 codes. In *Proc. of the IEEE/IFIP DSN'12*, Boston, MA, June 2012. 11

[93] Y. Zhu, P. Lee, L. Xiang, Y. Xu, and L. Gao. On the speedup of Single-Disk failure recovery in XOR-Coded storage systems: Theory and practice. In *Proc. of the IEEE MSST'12*, Pacific Grove, CA, April 2012. 11

Vita

Chentao Wu was born on June 15, 1982, in Xinzhou District, Wuhan City, Hubei Province, P.R. China. He is a Chinese citizen. He graduated from Xinzhou No.1 Middle School, Wuhan, China in 2000. He received his Bachelor of Engineering in Computer Science and Technology, his Master of Engineering in Software Engineering, both from Huazhong University of Science and Technology (HUST), Wuhan, China in 2004 and 2006, respectively. He is the recipient of the 2011-2012 VCU School of Engineering Outstanding Graduate Research Assistant Award.