A Survey of the Past, Present, and Future of Erasure Coding for Storage Systems

ZHIRONG SHEN and YUHUI CAI, Xiamen University, China
KEYUN CHENG and PATRICK P. C. LEE, The Chinese University of Hong Kong, China
XIAOLU LI and YUCHONG HU, Huazhong University of Science and Technology, China
JIWU SHU, Tsinghua University, China

Erasure coding is a known redundancy technique that has been popularly deployed in modern storage systems to protect against failures. By introducing a small portion of coded redundancy into data storage, erasure coding is shown to provide higher reliability guarantees than replication under the same storage overhead. Despite its storage efficiency, erasure coding incurs high performance overhead in repair and updates, and its reliability also depends on the amount of redundancy. How to resolve the tensions among storage efficiency, performance, and reliability has been the major research direction in the literature for decades.

In this paper, we present an in-depth survey of the past, present, and future of erasure coding in storage systems. We conduct our survey from a systems perspective, with an emphasis on how erasure coding is deployed in practical storage systems. Specifically, we first review the use of erasure coding in storage systems from both academia and industry, and state the challenges of deploying erasure coding in practice. We then review the topics of erasure coding in three aspects: (i) new erasure code constructions, (ii) algorithmic techniques for efficient erasure coding operations, and (iii) erasure coding for emerging architectures. Finally, we provide future research directions for erasure coding.

CCS Concepts: • **Computer systems organization** → **Reliability**; **Availability**; • **Information systems** → **RAID**; **Distributed storage**.

Additional Key Words and Phrases: Erasure coding, Distributed Storage Systems

1 INTRODUCTION

We have been witnessing an ever-increasing amount of data in the wild. A 2018 report from the International Data Corporation (IDC) forecasts that the total volume of data created, captured, and replicated across the globe will grow from 33 zettabytes in 2018 to 175 zettabytes by 2025 [158], and a more recent 2022 report from IDC further forecasts that the global data will reach 221 zettabytes by 2026 [20]. Statista has similar predictions for the exponential data growth and the global data will grow to 181 ZB, and the growth is higher than earlier expected due to the COVID-19 pandemic [181]. To manage tremendous amounts of data at scale, enterprises and organizations often keep data in large-scale distributed storage systems by aggregating the storage resources of multiple *nodes* (i.e., disks or servers).

As storage systems scale, node failures are inevitable. Failures can manifest in different types, ranging from *transient failures* (e.g., power loss, network disconnection, system upgrades, reboots, etc.) in which nodes and their stored data are temporarily unavailable for a relatively short time period, to *permanent failures* (e.g., disk crashes, sector errors, etc.) in which nodes and their stored data are permanently lost and need to be explicitly restored. To protect against (transient or permanent) failures, traditional distributed storage systems adopt *replication* due to its simplicity by

This work was supported in part by the National Key R&D Program of China (2022YFB4501200), Research Grants Council of Hong Kong (AoE/P-404/18), Major Research Plan of the National Natural Science Foundation of China (No. 92373114), Natural Science Foundation of China (No. 62072381, 62302175, and 62272185), and Natural Science Foundation of Fujian Province of China (No. 2023J06001). The corresponding author is Patrick P. C. Lee (pclee@cse.cuhk.edu.hk).
Authors' addresses: Zhirong Shen; Yuhui Cai, Xiamen University, Xiamen, China; Keyun Cheng; Patrick P. C. Lee, The Chinese University of Hong Kong, Hong Kong, China; Xiaolu Li; Yuchong Hu, Huazhong University of Science and Technology, Wuhan, China; Jiwu Shu, Tsinghua University, Beijing, China.

keeping multiple exact redundant copies of data across nodes for redundancy. However, the storage overhead of replication is significant and poses scalability concerns in the face of tremendous amounts of data to manage. Despite the continuous drops in disk prices, the unprecedented growth of data volume unavoidably inflates the operational costs of storage infrastructure, in terms of footprints, energy consumption, hardware/software maintenance, and personnel support.

Erasure coding is a low-cost redundancy mechanism that incurs significantly less redundancy than replication while preserving the same degree of reliability (measured in terms of mean-time-to-data-loss) [194]. At a high level, erasure coding works by encoding original data units to form extra redundant units, such that any subset of a sufficient number of data and redundant units can reconstruct the original data (see §2.1 for details). Erasure coding is reportedly deployed in production environments (§2.2).

While the theory of erasure coding has been well established for a few decades, there remain fundamental design trade-off issues regarding the practical deployment of erasure coding in modern storage systems. There are three dimensions of design trade-offs in erasure coding deployment: storage efficiency, performance, and fault tolerance. Specifically, erasure coding, while offering storage efficiency over replication, often incurs higher bandwidth and I/O costs than replication, especially in failure repair and data update operations. Also, to achieve higher performance and fault tolerance, erasure coding should be configured with high storage redundancy. How to balance the design trade-offs in erasure coding deployment is non-trivial, as it highly depends on the deployment scenarios and application workloads.

In this paper, we present an in-depth study of the past, present, and future of erasure coding in storage systems and review how the design trade-offs of erasure coding are addressed in the literature. Specifically, while erasure coding has also been extensively studied by the information theory community, we conduct our survey from a systems perspective, with a specific emphasis on the practical deployment of erasure coding. We start with reviewing the use of erasure coding in storage systems from both academia and industry, and state the challenges of deploying erasure coding in practice. We focus on three aspects:

- *New erasure code constructions:* We discuss the new erasure code constructions that are designed with performance guarantees on erasure coding operations (e.g., encoding/decoding, repair, and updates). Such constructions include: XOR-based erasure codes, regenerating codes, locally repairable codes, piggybacking codes, and sector-disk codes.
- *Algorithmic techniques for erasure codes:* We discuss the new algorithmic techniques that are designed to operate on existing erasure codes and improve the performance of erasure coding operations. Such algorithmic techniques include: acceleration algorithms for encoding/decoding operations, repair-efficient algorithms, update-efficient algorithms, consensus algorithms, change of erasure coding parameters, and reliability evaluation.
- *Erasure coding for emerging architectures:* We discuss the erasure coding deployment in fast storage devices (e.g., flash memories, DRAM-only storage, and disaggregated memory), programmable switches, and edge-cloud storage.

Prior surveys. We are aware of a few surveys on erasure coding. Plank and Huang present a tutorial for erasure coding and its applications at the FAST conference in 2013 [144]. Dimakis et al. [33] present a survey of regenerating codes for improving the repair performance of storage systems. Several surveys (e.g., Li and Li [99], Datta and Oggier [30], Liu and Ogger [119], Balaji et al. [13], and Ramkumar et al. [149]) study advanced erasure code constructions and specifically focus on two families of erasure codes: regenerating codes and locally repairable codes.

However, the above surveys are either published more than a decade ago [30, 32, 99, 144] or only focus on erasure code constructions [13, 30, 33, 99, 119, 149]. They neither address the algorithmic

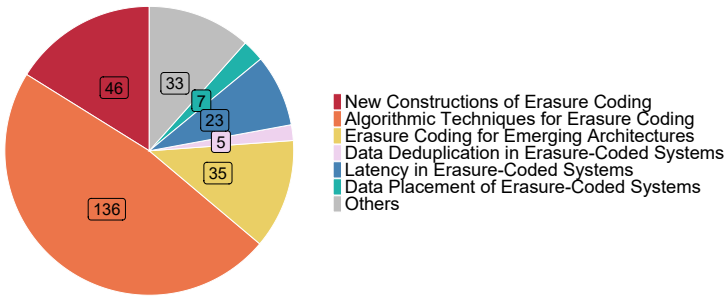


Fig. 1. Number of papers under different research topics.

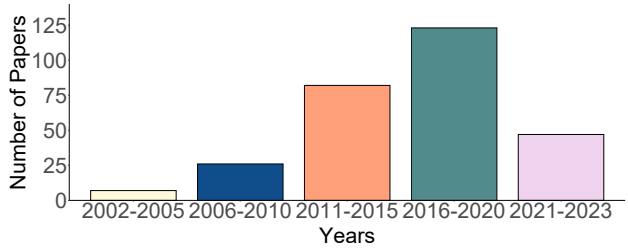


Fig. 2. Number of papers in erasure coding published by years.

techniques that apply to general erasure codes, nor study the erasure coding deployment in different storage architectures. We fill the void by providing a more up-to-date and comprehensive survey on erasure coding for storage systems from a systems perspective.

Analysis of research topics: To help us prepare this survey, we first analyze the state-of-the-art research topics on erasure-coded storage. We search the papers on the DBLP database [97] using the keyword ''erasure code storage''. The search results include the papers whose titles include the phrases ''erasure code storage'' and ''erasure-coded storage''. We focus on the papers that are published from 2002 to August 2023, and collected 285 papers in total.

We first analyze the research topics of the papers. Figure 1 provides a breakdown of the collected papers by different topics. We observe that 76.1% of all collected papers can be classified into three topics: (i) erasure coding constructions, (ii) algorithmic techniques for erasure coding (e.g., the algorithms for accelerating the encoding, repair, and update operations), and (iii) erasure coding for emerging architectures (e.g., in-memory storage, disaggregated memory, and edge-cloud storage). For the remaining papers, they have different research topics, such as deduplication in erasure-coded storage, read latency reduction in erasure-coded storage, and placement strategies in erasure-coded storage.

We further analyze the publication years of all collected papers. Figure 2 shows the number of papers versus different groups. We observe that the number of papers has significantly increased since 2002, implying that erasure coding has received significant attention in storage systems. We conjecture two possible reasons. First, with the development of network protocols and storage devices, storage architectures have been evolving to meet the new demands in data storage. Examples include disk arrays (which provide massive storage with parallel access), in-memory storage (which bridges the performance gap between high-performance networks and external

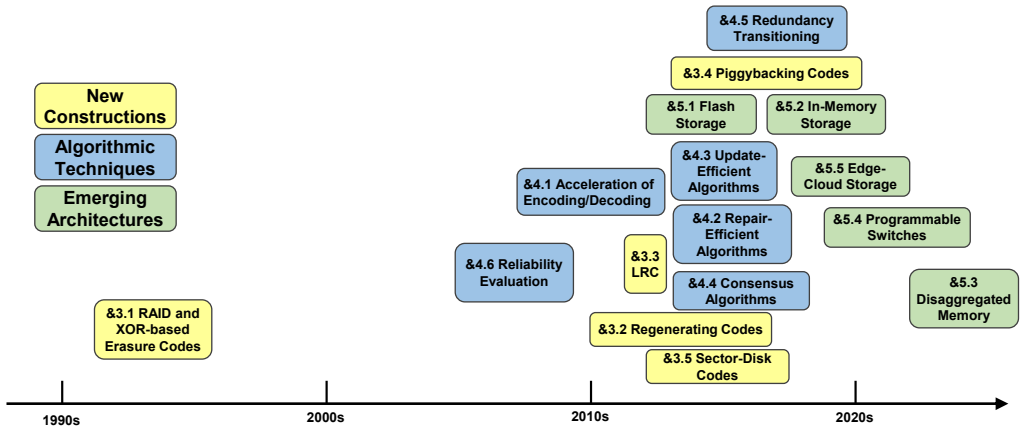


Fig. 3. Research trend of erasure coding in storage systems. We put a research topic in a timeframe based on when the topic first emerges and is studied by a number of papers.

storage), and disaggregated memory (which enables independent scaling of computing and memory resources). The changes in storage architectures also create new design considerations for data reliability guarantees, such as repair performance, memory efficiency, and fault tolerance, thereby expanding the design space of erasure coding. Second, the scale of storage systems continues to grow to accommodate the ever-increasing data volume, making failures prevalent rather than exceptional. Erasure coding is considered a promising means to guarantee data reliability with much less storage overhead than replication, and hence is extensively deployed in modern storage systems.

In this paper, we examine the representative papers in the following three main research topics in erasure coding: new erasure coding constructions, algorithmic techniques for erasure coding, and erasure coding for emerging architectures. Figure 3 further provides an overview of the specific sub-topics in the three main topics over time. We observe that new research topics emerge after the year 2010, which may be attributed to (i) the increasing significance of distributed storage and its performance and fault tolerance requirements (e.g., regenerating codes and locally repairable codes) as well as (ii) the emergence of new hardware (e.g., flash memory and programmable switches) and new architectures (e.g., in-memory storage, disaggregated memory, and edge-cloud storage).

Paper organization. The rest of the paper proceeds as follows. In §2, we introduce the basics of erasure coding and the adoption of erasure coding in distributed storage systems. We also highlight the challenges of erasure coding deployment to motivate our survey. In §3, we study state-of-the-art erasure code constructions. In §4, we study algorithmic techniques for erasure coding. In §5, we study erasure coding deployment for different types of architectures. In §6, we present the open problems and research directions that may advocate the future development of erasure coding for storage systems. Finally, in §7, we conclude this paper.

2 BACKGROUND OF ERASURE CODING

In this section, we provide an overview of erasure coding (§2.1). We elaborate on the adoption of erasure coding (§2.2), and finally state the challenges of erasure coding deployment (§2.3).

2.1 Basics of Erasure Coding

Erasure coding is a low-cost redundancy mechanism that generates minimal data redundancy through the mathematical computation of existing data being stored (as opposed to making multiple exact copies of data as in replication), so as to provide fault tolerance for storage systems. To apply erasure coding in storage systems, we construct an erasure code based on configurable parameters. There are many constructions of erasure codes proposed in the literature, among which Reed-Solomon (RS) codes [157] are the most widely deployed family of erasure codes used in production storage systems (§2.2). In this section, we use RS codes to introduce the basics of erasure coding, including notations, definitions, and key concepts, from a systems perspective.

Specifically, an RS code is constructed with two configurable positive integers k and n, where $n > k$. An (n, k) RS code divides the original (uncoded) data being stored into k fixed-size *chunks*. It encodes the k uncoded chunks to form n coded chunks of the same size, such that *any* k out of the n coded chunks can be decoded to the original k uncoded chunks. We refer to the collection of n coded chunks as a *stripe*. For scalable storage, a storage system typically stores multiple stripes, all of which are independently encoded and decoded. The n coded blocks of each stripe are distributed across n different nodes in a storage system to tolerate any $n - k$ node failures.

RS codes have several key properties that make them popular in practice.

- RS codes are *maximum distance separable (MDS)*, meaning that RS codes can reconstruct the original data with any k out of n chunks with the minimum storage redundancy (i.e., n/k times the original data size); in other words, RS codes are *storage-optimal*, and no other erasure codes can support such data reconstruction with less redundancy than RS codes.
- RS codes are *general*, meaning that the coding parameters n and k can be arbitrarily configured in practical deployment (provided that $n > k$). Note that n is typically upper-bounded due to the arithmetic constraint (see below), but such a constraint has a limited practical impact.
- RS codes can be configured in *systematic* form, meaning that k of the n coded chunks of a stripe are identical to the k uncoded chunks. In other words, the original data can be directly read from a stripe without decoding, thereby preserving the read performance in normal operations. In systematic form, we refer to the k uncoded chunks as *data chunks*, while the $n - k$ coded chunks as *parity chunks*.

We elaborate on the mathematical properties of RS codes, assuming that they are in systematic form. Given a stripe of an (n, k) RS code, let $D_0, D_1, \cdots, D_{k-1}$ be the k data chunks. For encoding, each parity chunk P_j, where $0 \leq j \leq n - k - 1$, is computed as a linear combination of the k data chunks of the same stripe, given by $P_j = \sum_{i=0}^{k-1} \alpha_{i,j} D_i$, for some coding coefficients $\alpha_{i,j}$'s (where $0 \leq i \leq k - 1$ and $0 \leq j \leq n - k - 1$) specified by the erasure code construction. For decoding, each data chunk can be computed as a linear combination of any k out of the n data and parity chunks of the same stripe with a different set of coding coefficients based on Gaussian elimination or matrix inversion [144]. All additions and multiplications are performed in the Galois Field GF(2^w) over w-bit *words*, where GF(2^w) represents a finite field with 2^w numbers from 0 to $2^w - 1$, and w represents the word size and is often selected as a power of two (e.g., $w = 4$, 8, 16, or 32). Each word is a basic unit for data manipulation in encoding and decoding operations. In particular, an addition is equivalent to bitwise XOR, a multiplication is performed over polynomial representations, and the arithmetic operations over the Galois Field are *closed* (i.e., the arithmetic outputs remain in the field). The details of the Galois Field operations are beyond the scope of this paper, and we refer readers to the previous papers on Galois Field operations for erasure coding [48, 143, 144, 195].

Note that the additions are *associative* (i.e., the order of additions is arbitrary). RS codes require that $n \leq 2^w$ [144]. A typical parameter of w is $w = 8$ (i.e., a word is equivalent to a single byte), implying that $n \leq 256$. Nevertheless, the stripe size n is often configured to be much less than 256

nodes to maintain repair efficiency (§4.2), so the constraint does not affect the practical deployment; recall that a large-scale storage system may still contain thousands of nodes that store multiple stripes of chunks. To support efficient encoding and decoding in practice, each chunk is partitioned into multiple w-bit words, such that the encoding and decoding operations are applied to the words at the same offset of the chunks of a stripe.

We briefly discuss the terminologies for erasure coding used in coding theory. A word can be referred to as a *symbol* or a *packet*. A stripe is also referred to as a *codeword*. The coding rate, k/n, is often used to quantify the storage redundancy of an erasure code (i.e., a higher coding rate implies lower storage redundancy).

2.2 Adoption of Erasure Coding

Erasure coding has been extensively used in storage systems. We review the adoption of erasure coding in RAID and the distributed storage systems proposed in academia dated back in the 1990s and 2000s. We also review the recent deployment of erasure coding in production storage systems reported since the 2010s.

RAID. RAID (Redundant Array of Independent Disks) is first proposed by Patterson et al. in 1988 [138] to form an array of multiple disk drives for increased capacity, scalability, and fault tolerance. In particular, for fault tolerance, RAID Level 5 (or RAID-5 in short) introduces a parity chunk into each stripe, so as to tolerate any single-disk failure, and rotates the parity chunks of all stripes across disk drives for load-balanced parity updates. RAID-5 can be viewed as a special form of erasure coding, in which $n - k = 1$, and significantly saves the storage overhead of replication in RAID Level 1 (a.k.a. mirroring). However, RAID-5 only provides single-disk fault tolerance and remains susceptible to data loss in a larger disk array. Thus, many follow-up studies (especially in the late 1990s and early 2000s) examine RAID-based codes with higher fault tolerance (see §3.1 for details).

Distributed storage systems from academia. In the 2000s, academics propose various wide-area or clustered distributed storage systems and apply erasure coding for fault tolerance. To name a few, OceanStore [92] stores objects in both active and archival forms in a geo-distributed setting, and employs erasure coding for archived objects with a high redundancy ratio (e.g., $n = 2k$). TotalRecall [15] stores small files with replication and large files with erasure coding in peer-to-peer networks; for erasure-coded chunks, it lazily repairs the failed chunks until at least a threshold number of chunks are unavailable. DHash++ [29] employs erasure coding in a distributed hash table (DHT) network and addresses the lookup and write performance of erasure-coded data. Glacier [52] employs erasure coding in decentralized storage environments and generates a sufficient degree of redundancy that can tolerate correlated failures with durability guarantees. Ursa Minor [7] considers single-data-center clustered-based environments and applies per-object replication or erasure coding for different data types. IrisStore [131] uses a failure size distribution model to determine the appropriate erasure coding parameters. SafeStore [90] applies hierarchical erasure coding across and within storage service providers and provides an interface for clients to configure the redundancy within a storage service provider.

Production storage systems. Erasure coding is reportedly deployed in production storage systems hosted by enterprises. Table 1 shows the examples of erasure coding deployment in production and the erasure coding parameters (n, k) being reported. All storage systems, except Microsoft Azure, employ RS codes, while Azure employs Local Reconstruction Codes (§3.3). Note that the typical parameters of (n, k) are limited to no more than 20 to limit the repair penalty (§2.3). Also, the number of tolerable failures, $n - k$, is set to three or four, and the redundancy n/k is less than two (i.e., the redundancy is less than that of replication).

Table 1. Erasure coding deployment in production (the table is reported in [61]). All storage systems (except Azure) are based on RS codes [157], while Azure is based on Local Reconstruction Codes [66].

Systems	(n, k)	Redundancy (n/k)
Google Colossus [40]	(9,6)	1.50
Quantcast File System [133]	(9,6)	1.50
Hadoop Distributed File System [3]	(9,6)	1.50
Baidu Atlas [93]	(12,8)	1.50
Facebook f4 [129]	(14,10)	1.40
Yahoo Cloud Object Store [6]	(11,8)	1.38
Microsoft Windows Azure Storage [66]	(16,12)	1.33
Tencent Ultra-Cold Storage [5]	(12,10)	1.20
Pelican [14]	(18,15)	1.20
Backblaze Vaults [1]	(20,17)	1.18

2.3 Challenges of Erasure Coding Deployment

Erasure coding, while significantly reducing the storage overhead of replication, incurs higher performance overhead than replication. Here, we discuss the performance overhead in three aspects: coding computations, failure repair, and data updates.

Coding computations. Erasure coding encodes storage redundancy by performing arithmetic operations on the original data, and decodes the original data by performing arithmetic operations on the available coded chunks. The encoding/decoding operations incur computational overhead; in contrast, replication directly forms redundant data copies without arithmetic operations. Such computational overhead is insignificant in distributed storage systems whose performance bottlenecks are in network transmissions or I/O accesses as opposed to computations (e.g., geo-distributed data centers), but it becomes non-negligible for high-speed networked environments or fast storage devices. Thus, existing studies construct erasure codes that have simple operations (e.g., bitwise XOR operations only) (§3.1), or design acceleration techniques for encoding/decoding operations (§4.1).

Failure repair. When a node fails and its chunks are no longer accessible, repair operations should be carried out to reconstruct the failed chunks to maintain data availability. There are two types of repair operations: (i) *degraded reads*, which reconstruct any failed data chunk to which reads are issued, and (ii) *full-node recovery*, which reconstructs all lost chunks of a permanently failed node into a newly non-failed node. In both cases, a repair operation for a failed chunk under erasure coding needs to retrieve a sufficient number of available chunks of the same stripe for decoding (e.g., in an (n, k) RS code, k available chunks of the same stripe need to be retrieved). This incurs bandwidth and I/O amplification; in contrast, replication only retrieves another available redundant copy for reconstruction. Thus, existing studies construct erasure codes (e.g., regenerating codes, locally repairable codes, piggybacking codes) that aim to mitigate the bandwidth and I/Os incurred during a repair operation (§3.2, §3.3, and §3.5), or design repair-efficient techniques that accelerate repair operations (§4.2).

Data updates. For systematic erasure codes, if any data chunk is updated, the parity chunks of the same stripe also need to be updated for consistency. There are two types of updates: (i) *full-stripe updates*, in which all data chunks of a stripe are updated, and (ii) *partial-stripe updates*, in which some but not all data chunks of a stripe are updated. For full-stripe updates, a storage system simply re-computes the parity chunks of the same stripe based on the latest data chunks and writes the new stripe in its entirety. For partial-stripe updates, a storage system can perform one of the following two update approaches based on the linear property of practical erasure codes [182]:

(a) *reconstruct-writes*, which read all non-updated chunks of the stripe, compute the new parity chunks based on the newly updated chunks and the non-updated chunks that have been read, and write the newly updated chunks and the new parity chunks, and (b) *read-modify-writes*, which read the existing data chunks to be updated and all existing parity chunks, compute the deltas (i.e., content differences) of the newly updated chunks and existing data chunks, compute the new parity chunks by applying the deltas to the existing parity chunks, and write all newly updated data chunks and new parity chunks. To elaborate on read-modify-writes under (n, k) systematic RS codes, suppose that an existing data chunk D_i is updated to a new data chunk D_i' for some i (where $0 \leq i \leq k-1$). Each new parity chunk P_j' can be computed from the existing parity chunk P_j with the delta (i.e., $D_i' - D_i$) (for $0 \leq j \leq n-k-1$) as:

$$P_j' = P_j + \alpha_{i,j}(D_i' - D_i) \text{ for } 0 \leq j \leq n-k-1. \quad (1)$$

In partial-stripe updates, there are extra I/Os of reading existing chunks; in contrast, in replication, the updates to a data copy can be directly written to other redundant copies. Thus, existing studies construct erasure codes that aim to minimize the I/Os in updates (§3.1), or design update-efficient techniques that mitigate the I/Os in updates for general erasure codes (§4.3).

3 NEW CONSTRUCTIONS OF ERASURE CODING

In this section, we study the new constructions of erasure coding, with certain provable guarantees on performance and fault tolerance. We review these constructions based on how they evolve over time, and such constructions have been implemented and evaluated in practical storage systems. Specifically, we start with studying XOR-based erasure codes for RAID (§3.1), which are popularly studied in the late 1990s and early 2000s for high-performance, fault-tolerant disk arrays. We then study regenerating codes (§3.2) and locally repairable codes (§3.3), both of which are studied around the year 2010 for distributed storage to reduce the repair bandwidth and I/O. We next review sector-disk codes (§3.4), which address the combination of failures with different granularities. Furthermore, we study piggybacking codes (§3.5), which reduce the repair bandwidth and I/O while maintaining optimal storage efficiency. Finally, we provide a trade-off analysis between the repair overhead and the storage overhead for some of the constructions (§3.6).

Table 2 also shows the major acronyms and descriptions regarding the representative erasure codes that are discussed in this paper (e.g., RS codes, RAID, MSR/MBR codes, LRCs, sector-disk codes, and piggybacking codes) and their properties (e.g., MDS and non-MDS).

3.1 XOR-based Erasure Codes for RAID

Recall from §2.2 that the seminal paper of RAID [138] specifies how to tolerate a single-disk failure in a disk array by adding a parity chunk to each stripe, yet a single-disk fault tolerance is insufficient for protecting against data loss in large-scale storage systems.

To improve the fault tolerance of RAID, several studies focus on RAID-6 codes, which provide double-disk fault tolerance (i.e., $n-k=2$), or highly fault-tolerant RAID-based codes that can tolerate more than two disk-drive failures (i.e., $n-k>2$). One common feature of all such RAID-based codes is that their encoding and decoding operations are based on bitwise XOR operations only (i.e., no Galois Field arithmetic), which is critical to maintain high performance in RAID deployment.

RAID-6 codes. EVENODD [16] is the first RAID-6 code that is storage-optimal (i.e., MDS) and has the optimal encoding complexity (i.e., the number of XOR operations for parity construction is minimum). It organizes each stripe as a $(p-1) \times (p+2)$ two-dimensional array of elements for some prime number $p > 2$, where each column corresponds to a chunk. Let $b_{i,j}$ (where $0 \leq i \leq p-2$ and $0 \leq j \leq p+1$) be an element in the array. The first p columns (i.e., columns $0, 1, \cdots, p-1$)

Table 2. Representative erasure codes considered in the paper.

Codes / Properties	Descriptions
Maximum Distance Separable (MDS)	A property that an erasure code has the minimum storage redundancy (i.e., storage-optimal) for a tolerable number of failures; in other words, no erasure code can have less storage redundancy to tolerate the same number of failures.
Reed-Solomon (RS) codes [157]	A family of MDS erasure codes constructed over the Galois Field.
Redundant Array of Independent Disks (RAID) [138]	An array of multiple disk drives organized for high capacity, scalable performance, and fault tolerance. Popular coding schemes for RAID are RAID-5 and RAID-6, which provide single-disk and double-disk fault tolerance, respectively. Both RAID-5 and RAID-6 are MDS.
Minimum-Storage Regenerating (MSR) codes [32]	A family of regenerating codes that minimize the repair bandwidth subject to the minimum storage overhead. MSR codes are MDS.
Minimum-Bandwidth Regenerating (MBR) codes [32]	A family of regenerating codes that minimize the storage overhead subject to the minimum repair bandwidth. MBR codes are non-MDS and incur higher storage overhead to maintain fault tolerance.
Locally Repairable Codes (LRCs) [66, 87, 178]	Erasure codes that support local repair operations through the addition of local parity chunks within a stripe. LRCs are non-MDS and trade storage efficiency for repair efficiency.
Sector-Disk codes [18, 142]	Erasure codes that are constructed in the form of a two-dimensional array to provide fault tolerance against mixed sector and disk failures. SD codes are non-MDS, but are also referred to as partial-MDS as each row is encoded with an MDS code.
Piggybacking codes [156]	Erasure codes that couple the information from some sub-stripes to other sub-stripes so as to mitigate the repair bandwidth and I/O. Piggybacking codes can build on RS codes [157] and are MDS.

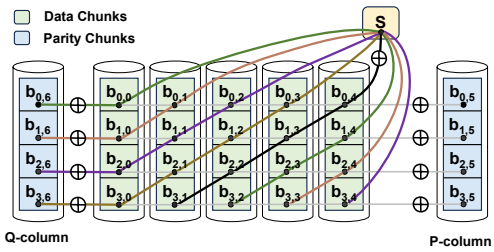


Fig. 4. EVENODD (where $p = 5$).

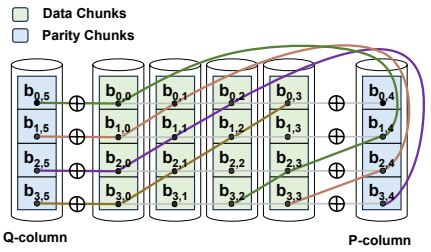


Fig. 5. RDP (where $p = 5$).

store data chunks, while the last two columns (i.e., columns p and $p + 1$, also referred to as the *P-column* and *Q-column*, respectively) store parity chunks. Figure 4 shows the encoding operations for $p = 5$. Each parity element in the P-column (i.e., column p) is encoded by XOR-summing all data elements in the same row:

$$b_{i,p} = \sum_{j=0}^{p-1} b_{i,j}, \text{ where } 0 \leq i \leq p-2.$$

Here, $\sum$ represents the XOR-summation (§2.1). For the parity elements in the Q-column (i.e., column $p + 1$), EVENODD first computes an *adjuster* S by XOR-summing data elements along the diagonal direction, followed by XOR-summing the adjuster and other data elements in other

diagonal directions (Figure 4):

$$b_{i,p+1} = S + \sum_{j=0, j\neq i+1}^{p-1} b_{(i-j) \bmod p, j}, \text{ where } S = \sum_{j=1}^{p-1} b_{p-1-j,j}.$$

RDP [28] improves the encoding performance by reusing the parity elements of the P-column to compute the parity elements of the Q-column. Specifically, it organizes each stripe as a $(p-1)\times(p+1)$ two-dimensional array of elements. Each parity element in the P-column is still encoded by XOR-summing all data elements in the same row (as in EVENODD), while each parity element in the Q-column is encoded by XOR-summing a parity element in the P-column and data elements along the diagonal direction. Figure 5 shows an example of RDP for $p = 5$.

Both EVENODD and RDP can support any number of data columns with the *shortening* technique. For example, for EVENODD, given k data columns where k is non-prime, we choose the next smallest prime p' that is larger than k. We keep zero elements in the additional $p' - k$ columns (which are not actually stored) and construct the codes based on p'.

Both EVENODD and RDP are *horizontal* codes, in which parity elements are stored in dedicated columns. They are sub-optimal in data updates, as the update of a data element triggers the updates of more than two parity elements. X-Code [215] is a *vertical* code by placing parity elements in two rows that span across all columns. It generates parity elements by XOR-summing data elements along the diagonal and anti-diagonal directions. It achieves optimal data updates, meaning that each data update triggers the updates of only two parity elements (which is minimum).

Several follow-up RAID-6 codes have been proposed. Liberation Codes [141] achieve optimal or near-optimal performance in encoding, updates, and decoding, and the code implementation is open-sourced. HDP Code [201] generates parity elements in horizontal-diagonal and anti-diagonal directions and aims for I/O load balancing in updates. Note that HDP Code does not achieve optimal updates. H-Code [202] aims to optimize partial-stripe updates and ensures that any two continuous data elements share a common parity element, such that any update to two continuous data elements only needs to update three associated parity elements. D-Code [43] is a vertical code (similar to X-Code) that achieves load balancing and reduces the I/O costs in reads by increasing the likelihood of having continuous data elements share common parity elements. HV Code [170] reduces the length of each *parity chain* (i.e., a collection of a parity element and its associated data elements for encoding) and achieves the optimal encoding and update complexities.

Highly fault-tolerant RAID-based codes. Some XOR-based erasure codes aim to provide higher fault tolerance beyond RAID-6. Note that RS codes can be implemented as Cauchy-RS codes [19] with XOR operations only and can tolerate a general number of failures. To preserve the performance benefits of RAID-6 codes, STAR [67] and RTP [44] extend EVENODD and RDP with triple-disk fault tolerance, respectively. TIP-Code [235] supports triple-disk fault tolerance and achieves optimal updates. The generalized EVENODD codes [17] extend EVENODD to tolerate multiple failures.

Some code constructions trade storage efficiency for high fault tolerance, and they are all non-MDS. WEAVER codes [53] use a bipartite graph in code construction to determine how each parity element is encoded by a fixed number of data elements. They can tolerate up to 12 disk failures, but incur high redundancy (at least 2×). HoVer codes [54] generate parity elements along both the horizontal and vertical directions and can tolerate up to four disk failures. GRID codes [102] construct multi-dimensional grids that can tolerate up to 15 and even higher failures. CORE [37] has similar ideas to GRID codes by generating parity elements in a grid-like structure and is implemented on Hadoop HDFS. PSG-Codes [108] partition disks into groups and generate parity elements with short parity chains, and are shown to tolerate up to 12 disk failures. Alpha Entanglement codes [38] combine multiple sets of independent data elements to generate parity

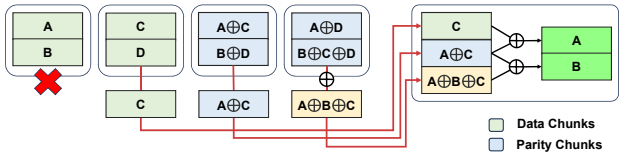


Fig. 6. Regenerating code (where $(n, k) = (4, 2)$). To repair the failed chunk (with two sub-chunks), each node transfers one coded sub-chunk. The repair bandwidth is 1.5× chunks, as opposed to $k = 2$ chunks in conventional repair.

elements, so as to create an exponential number of paths for data reconstruction for high data availability against disasters. RAIDP [161] combines replication and erasure coding in construction by keeping two replicas in different disks and generating an XOR parity for the chunks in each disk, so as to maintain high repair efficiency while reducing the storage overhead of replication.

3.2 Regenerating Codes

Recall from §2.3 that erasure coding amplifies bandwidth and I/Os in failure repair operations, thereby degrading the overall repair performance, especially in distributed storage systems where the performance bottleneck lies in network transmissions and disk I/Os.

Regenerating codes [32] have been proposed to minimize the repair bandwidth (i.e., the amount of network traffic transferred during a repair operation) for repairing a single-chunk failure in distributed storage systems. The key observation is that for a single stripe, single-chunk failures are much more common than concurrent-chunk failures in practice (e.g., more than 98% of failure events of a stripe are single-chunk failures in a Facebook's warehouse cluster [153]). Regenerating codes build on the theory of network coding [9], in which the available nodes encode their locally stored data and send the encoded data for reconstruction. They also perform *sub-packetization*, in which each chunk is partitioned into smaller sub-chunks, where the encoding and repair operations are performed at the sub-chunk granularity. Figure 6 shows an example of how a single-chunk repair operation is done in regenerating codes for $(n, k) = (4, 2)$. The seminal paper by Dimakis et al. [32] proves via the information flow graph analysis the optimal trade-off curve between storage overhead and repair bandwidth, and further characterizes the codes for two extreme points: (i) *minimum-storage regenerating* (MSR) codes, which minimize the repair bandwidth subject to the minimum storage overhead, and (ii) *minimum-bandwidth regenerating* (MBR) codes, which minimize the storage overhead subject to the minimum repair bandwidth. In particular, MSR codes are MDS and achieve the same storage optimality as RS codes. For example, for $n - k = 2$, MSR codes can reduce the repair bandwidth of RS codes for repairing a single-node failure by close to 50%.

Several follow-up studies propose new regenerating code constructions. Some theoretical studies propose regenerating codes based on interference alignment [163, 177] and product-matrix operations [155]. On the applied side, some studies implement non-systematic MSR codes [24] and network codes [10] for archival storage. Several studies design and implement systematic MSR codes in distributed storage systems (e.g., Ceph and Hadoop HDFS), including PM-RBT codes [152], Butterfly codes [135], HashTag codes [91], and Clay codes [186]. One key feature of the implemented MSR codes [91, 135, 186] is that they are *access-optimal*, meaning that both the amounts of repair bandwidth and disk I/Os are minimized. Some of the MSR codes have restrictions on erasure coding parameters (e.g., $n - k = 2$ for FMSR codes and Butterfly codes, and $n \geq 2k - 1$ for PM-RBT codes), yet Clay codes can support general (n, k) and are the state-of-the-art MSR codes.

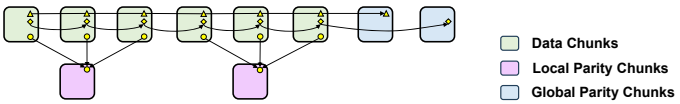


Fig. 7. Azure-LRC (where $(k, l, g) = (6, 2, 2)$).

Regenerating codes do not consider the hierarchical nature of data centers, in which cross-rack bandwidth is more limited than inner-rack bandwidth (note that racks can be mapped to regions in geo-distributed data centers). Double Regenerating Codes [62] are MSR codes that minimize the cross-rack repair bandwidth for repairing a single-node failure, and they have been implemented on Hadoop HDFS.

3.3 Locally Repairable Codes

Locally repairable codes (LRCs) are another popular erasure code construction that trades storage efficiency for reduced repair bandwidth and I/Os by supporting local repair operations through the addition of local parity chunks to a subset of chunks within a stripe (note that LRCs are also used as an acronym for Local Reconstruction Codes [66] and Local Recoverable Codes [178]). Local Reconstruction Codes [66] (and the predecessor Pyramid Codes [65]) are the family of locally repairable codes that are deployed in Microsoft Windows Azure storage (referred to as *Azure-LRC*). Specifically, Azure-LRC is configured by three parameters (k, l, g), with the stripe size $n = k + l + g$. Each stripe organizes k data chunks into l local groups of $\frac{k}{l}$ data chunks each (assuming that k is divisible by l). It generates a *local parity chunk* for each local group based on the data chunks within the local group, and further generates g *global parity chunks* based on all k data chunks of the stripe. It satisfies the Maximally Recoverable property [66] and can tolerate any $g + 1$ chunk failures within a stripe. To repair any failed chunk in a local group, it retrieves the $\frac{k}{l}$ chunks of the same local group (instead of reading k available chunks of the same stripe as in RS codes), thereby reducing the repair bandwidth and I/Os. Figure 7 shows an example of Azure-LRC, where $(k, l, g) = (6, 2, 2)$.

However, Azure-LRC only provides locality for the data chunks and local parity chunks, while repairing any of the failed global parity chunks still needs to retrieve k available chunks of the same stripe for reconstruction. Xorbas [162] adds an *implied* local parity chunk (which is not actually stored) to the group of global parity chunks so that each of the global parity chunks can be locally repaired. It focuses on finding the encoding coefficients that maximize the fault tolerance, and is implemented on Hadoop HDFS. Simple Regenerating Codes [137] combines MDS codes and locally decodable parity chunks for efficient repair. Kolosov et al. [87] systematically analyze the design trade-offs of Azure-LRC, Xorbas, and Optimal-LRCs [178] in terms of the average degraded read cost and the normalized repair cost.

Recent studies also consider *wide-stripe* erasure coding, in which k is very large and $n - k$ is small (e.g., $k \approx 100$ and $n - k = 3$ or 4). Wide-stripe erasure coding aims for extreme storage efficiency (with almost no redundancy overhead), but it also incurs high repair bandwidth and I/Os. Locally repairable codes are suitable candidates for constructing wide-stripe erasure codes due to their repair efficiency. ECWide [61] proposes combined locality, which combines parity locality (from Azure-LRC) and topology locality (due to rack-local repair operations), and uses Azure-LRC in rack-based data centers to reduce the repair bandwidth and I/Os. XHR-Code [220] follows the notion of combined locality by combining XOR codes, Hitchhiker code [154] (§3.5), and RS codes for wide stripes. Kadekodi et al. [85] state that wide-stripe locally repairable codes are deployed in

a Google storage cluster, and propose Uniform Cauchy Locally Repairable Codes that improve the reliability of existing locally repairable codes in various measures.

3.4 Sector-Disk Codes

Storage systems suffer from both *disk failures* and *sector failures*, where disk failures imply the loss of all data in an entire disk, while sector failures imply the loss of data in a specific sector of a disk (of a flash block of an SSD). *Latent sector errors* are commonly found on disks, where a corrupted sector is only detected when it is subsequently accessed or when the disk is proactively scrubbed. Conventional erasure codes provide device-level redundancy against disk failures, but the amount of redundancy becomes an overkill for protecting against sector failures and significantly increases the storage overhead.

A family of erasure codes called *sector-disk codes* has been proposed to provide storage-efficient redundancy against a mixture of disk and sector failures. Partial-MDS (PMDS) codes [18] and Sector-Disk (SD) codes [142] are two classical sector-disk codes. They are typically configured with four parameters (n, m, s, r). Each stripe comprises n chunks with r sub-chunks (or sectors) each, in which m of the n chunks are local parity chunks for protecting against disk failures. The (local parity) sub-chunks in each row of the m local parity chunks are encoded from the $n - m$ data sub-chunks of the same row. Furthermore, each stripe generates s global parity sub-chunks based on all $n \cdot r$ sub-chunks. To differentiate, PMDS codes tolerate any m sub-chunk failures in a row plus any s additional sub-chunk failures of the same stripe, while SD codes tolerate any m disk failures plus any s additional sub-chunk failures of the same stripe. However, SD codes only support a limited number of configurations, and it is computationally expensive to find feasible configurations in implementation.

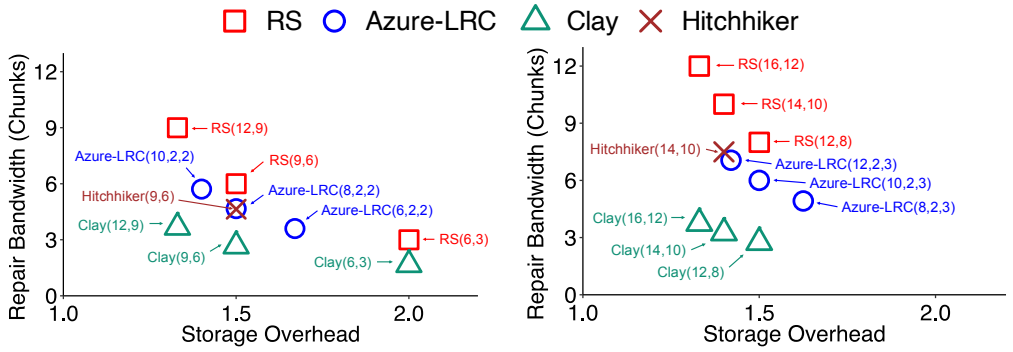
STAIR codes [100] address the computational inefficiency of SD codes, while providing storage efficiency and fault tolerance. The core idea of STAIR codes is to configure a vector of the number of tolerable sector failures across all chunks in a stripe. With the configured vector, STAIR codes encode the data chunks based on two systematic MDS codes in row and column directions, and repair a failed sub-chunk in a staircase manner.

While PMDS codes, SD codes, and STAIR codes address the mixture of disk and sector failures in a monolithic disk array, R-STAIR (recovery-oriented STAIR) codes [101] address the combination of rack and node failures in a hierarchical storage setting. R-STAIR codes extend STAIR codes to maintain rack-local parity chunks to mitigate the cross-rack repair bandwidth for a single-node repair operation, while preserving the generality of coding configurations as in STAIR codes.

3.5 Piggybacking Codes

While access-optimal MSR codes are MDS and minimize both repair bandwidth and I/Os, they introduce exponentially high sub-packetization overhead [91, 135, 186], which incurs non-sequential I/Os in accessing smaller sub-chunks. On the other hand, locally repairable codes, albeit being repair-efficient, are non-MDS and hence incur additional storage overhead. *Piggybacking codes* are a class of MDS codes that stand in the middle of the design space, in which they have smaller (yet sub-optimal) repair bandwidth and I/Os, while limiting the sub-packetization overhead. The core idea of constructing piggybacking codes is to operate on a limited number of sub-stripes of MDS codes (e.g., RS codes) and couple the information from some sub-stripes to other sub-stripes through some piggybacking functions. The repair of a chunk retrieves sub-chunks based on the coupled information from the corresponding sub-stripes.

Hitchhiker codes [154] are the state-of-the-art piggybacking codes that are implemented on Hadoop HDFS, by constructing a stripe from two sub-stripes of RS codes. They are shown to reduce the repair bandwidth and I/Os of RS codes by 25-45%. Rashmi et al. [156] formalize a piggybacking



(a) Tolerating three failures (b) Tolerating four failures

Fig. 8. Trade-off analysis between repair bandwidth and storage overhead subject to a tolerable number of failures.

framework and provide feasible constructions of piggybacking codes for general erasure coding parameters.

3.6 Trade-off Analysis

We have so far introduced different families of erasure codes. We now study their design trade-offs. Recall that there exist erasure coding deployment makes design trade-offs in three dimensions: storage efficiency, performance, and fault tolerance (§1). Here, we analyze the design trade-offs between repair bandwidth (which provides a measure of repair performance) and storage overhead, subject to a fixed tolerable number of failures, for some of the representative erasure codes that have been used in distributed storage systems with different configuration parameters.

Specifically, we focus on four erasure codes: (i) RS codes [157] with parameters (n, k) (§2.1), (ii) Azure-LRC [66] with parameters (k, l, g) (§3.3), (iii) Clay codes [186] with parameters (n, k) (§3.2), and (iv) Hitchhiker codes [154] with parameters (n, k) (§3.5). We consider Azure-LRC, Clay codes, and Hitchhiker codes as they are the representative constructions of LRCs, MSR codes, and piggybacking codes. We do not consider RAID and sector-disk codes as they are designed mainly for disk arrays instead of distributed storage systems.

We elaborate on our analysis methodology as follows. We fix the number of tolerable failures (three and four in our case) and analyze the trade-off between repair bandwidth and storage overhead for different parameters. We measure the storage overhead as $\frac{n}{k}$ for RS codes, Clay codes, and Hitchhiker codes, and as $\frac{k+l+g}{k}$ for Azure-LRC (as Azure-LRC can tolerate any $g+1$ chunk failures (§3.3), we set $n-k=g+1$). Also, we measure the repair bandwidth as the number of chunks retrieved to repair a lost chunk. For RS codes, the repair bandwidth is k chunks. For Azure-LRC, each of the data chunks and local parity chunks is repaired by retrieving $\frac{k}{l}$ chunks in the global group, while each global parity chunk is repaired by retrieving k chunks as in RS codes. Thus, the (average) repair bandwidth of Azure-LRC is $\frac{(k+l)k/l+gk}{k+l+g}$ chunks. For Clay codes, the (minimum) repair bandwidth is $\frac{n-1}{n-k}$ chunks. For Hitchhiker codes, we focus on $(n, k) = (9, 6)$ and $(n, k) = (14, 10)$, in which each data chunk is repaired by retrieving an average of 3.96 and 6.5 chunks, respectively, while each global parity chunk is repaired by retrieving k chunks as in RS codes. Thus, the (average) amounts of repair bandwidth of Hitchhiker codes under (9,6) and (14,10) are 4.64 and 7.5 chunks, respectively.

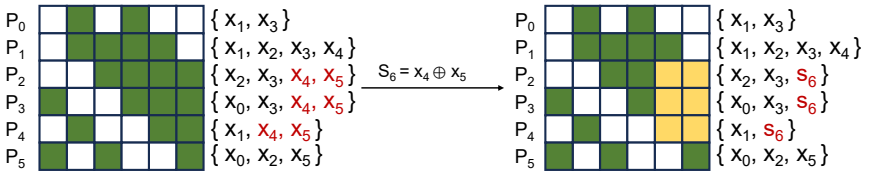


Fig. 9. XOR-scheduling in encoding [146]. The scheduling finds the common data chunks read for encoding (i.e., x_4 and x_5) and generates an intermediate chunk (i.e., s_6), so as to reduce XOR operations performed in encoding.

Figure 8 shows the trade-off results for three and four tolerable failures for different parameter combinations. We observe that RS codes have the highest repair bandwidth, Azure-LRC and Hitchhiker codes have less repair bandwidth with similar storage overhead, and Clay codes have the least repair bandwidth. Also, comparing Figures 8(a) and 8(b), the repair bandwidth is higher when the storage overhead is lower (and vice versa) under the same number of tolerable failures, while the repair bandwidth is higher when the number of tolerable failures is higher under the same storage overhead (e.g., RS(9,6) versus RS(16,12)). In addition, while Clay codes reach the Pareto optimality in repair bandwidth and storage overhead, they come with a higher sub-packetization level (i.e., with much smaller sub-chunks in data encoding and decoding), resulting in non-sequential I/Os in repair operations.

4 ALGORITHMIC TECHNIQUES FOR ERASURE CODING

In this section, we study the algorithmic techniques that are designed for existing erasure code constructions.

4.1 Acceleration of Encoding/Decoding Operations

Erasure coding generates redundancy through computations, thereby incurring non-negligible computational overhead in encoding and decoding operations. Extensive studies in the literature have studied the optimization techniques of encoding and decoding operations from different perspectives.

Performance evaluation. It is important to first understand the performance impact in erasure coding operations. Greenan et al. [48] study the performance of Galois Field implementation of GF(2^w) for $w \in \{4, 8, 16, 32\}$. Plank et al. [145] study the performance of five open-source erasure coding libraries in different architectures, and show the features and parameters that improve the performance. Zhou et al. [241] analyze existing encoding/decoding optimization techniques (e.g., optimizing the bitmatrix design, reducing the computational overhead, and cache-friendly scheduling) and study the performance gains when they are used together.

Scheduling. Some studies propose to schedule the encoding and decoding operations to reduce the number of XOR operations for better cache efficiency. Plank et al. [146] consider the scheduling of Cauchy-RS coding [19], which converts the finite field arithmetic into a sequence of XOR operations to generate parity chunks (§3.1). They propose to schedule the calculation of parity chunks and reuse the parity chunks that have already been computed as input to generate the remaining parity chunks, so as to minimize the number of XOR operations (Figure 9). Plank et al. [143] employ Intel SIMD Instructions for fast Galois Field arithmetic by optimizing the multiplication between the regions of bytes and constants, and the optimizations are included in the Jerasure version 2.0 library [76]. Luo et al. [122] observe that CPU cache efficiency greatly affects the encoding throughput. They study four XOR-scheduling algorithms that have different priorities on spatial locality and

temporal locality when accessing data and parity chunks, and show that the locality of data chunks is much more important than that of parity chunks in encoding. Uezato [184] optimizes XOR-based erasure coding using program optimization techniques, including grammar compression algorithms for reducing the number of XOR operations, deforestation for reducing memory access, and pebble game of program analysis for reducing cache misses. Li et al. [109] optimize encoding and decoding for asymmetric parity erasure codes (e.g., SD codes [142] and Azure-LRC [66]), where a stripe comprises different types of parity chunks with specific encoding rules, by partitioning encoding and decoding operations into sub-operations and processing them in parallel.

4.2 Repair-Efficient Algorithms

To reduce the repair penalty in erasure-coded storage, several studies propose new repair-efficient algorithms that apply to general erasure codes (e.g., RS codes) instead of proposing new erasure code constructions. While such algorithms cannot theoretically minimize the repair bandwidth and I/Os, they are empirically shown to improve the repair performance. We classify the representative studies related to repair-efficient algorithms into the following six sub-topics, in which these algorithms have been implemented and empirically evaluated. They include: (i) the repair algorithms that mitigate and balance I/Os, (ii) the repair algorithms that exploit transmission parallelism, (iii) the proactive repair algorithms that narrow the time window of data vulnerability, (iv) the repair algorithms that accelerate multi-chunk repair operations, (v) the repair algorithms that address the bandwidth differences in hierarchical data centers, and (vi) the read algorithms that leverage the repair capability of erasure coding.

Repair-efficient algorithms for RAID and XOR-based erasure codes. Several studies support fast repair mechanisms for RAID. Holland et al. [60] propose a parity declustering strategy by distributing RAID stripes to different subsets of disks in a large disk array, so as to parallelize repair operations across all disks. D-GRAID [130] employs selective metadata replication (i.e., replicating naming and system metadata structures) and fault-isolated data placement (i.e., storing semantically-related chunks within the same disk) to mitigate the interference to disks in repair operations. Hafner et al. [55] leverage the matrix theory to efficiently recover correlated or uncorrelated lost sectors for general XOR-based erasure codes. OI-RAID [191] proposes a two-layer RAID architecture by combining inner- and outer-layer RAID-5 codes with disk grouping to reduce the amount of disk reads and balance the repair loads. RAID+ [227] considers large disk enclosures (e.g., with 60 disk drives) and proposes deterministic and balanced data distribution based on mutually orthogonal Latin squares to balance the normal I/O and repair loads.

Several studies design I/O-efficient repair algorithms for XOR-based erasure codes by mitigating the I/Os (measured by the number of elements retrieved) in repair operations. Xiang et al. [211] propose a hybrid recovery approach for two RAID-6 codes, RDP and EVENODD, by exploiting both parity chunks to provably minimize the number of elements being read for repairing a single disk failure. Xu et al. [217] apply a similar hybrid recovery approach to optimize the repair of a single-disk failure for X-Code. Khan et al. [86] enumerate all possible decoding equations for the generator matrix of an XOR-based erasure code to minimize the number of elements being read for repairing a single-disk failure. Zhu et al. [243] propose a greedy algorithm to accelerate the search for an I/O-efficient repair algorithm. SIOR [174] further minimizes the disk seek operations in the repair operations.

Repair parallelization. Several studies propose efficient parallelization of repair algorithms in distributed storage systems to reduce the overall repair time. Degraded-first scheduling [105] launches degraded read tasks with higher priorities in MapReduce when running on erasure-coded Hadoop HDFS. PUSH [68] performs the repair in a pipelined manner at the chunk level

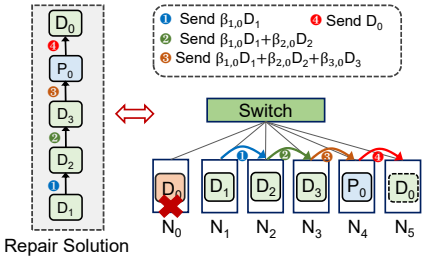


Fig. 10. PUSH [68].

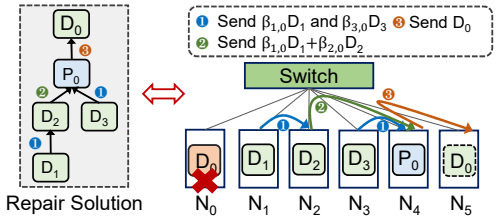


Fig. 11. PPR [126].

(Figure 10). Partial-Parallel-Repair (PPR) [126] partitions the repair of a single chunk into multiple sub-operations and parallelizes their executions across multiple nodes (Figure 11). ECPipe [106] partitions a chunk into small-sized sub-chunks and parallelizes the repair of sub-chunks in a pipelined manner; its extended version [114] further extends repair pipelining for rack-based data centers and the repair of multiple failed chunks. Follow-up studies (e.g., Parallel Pipeline Tree (PPT) [12], SMFRepair [238], RepairBoost [116], and PivotRepair [224]) propose parallel repair strategies for heterogeneous environments whose link bandwidth varies across nodes.

Implementing new repair-efficient algorithms in distributed storage systems requires re-engineering of the existing I/O workflows and is non-trivial. OpenEC [113] proposes a middleware framework that can be readily deployed atop existing distributed storage systems with limited code changes. It abstracts the flows of erasure coding operations as directed acyclic graphs, which can be configured to support PPR [126] and repair pipelining [106]. It is later extended to support the elastic transformation of a base erasure code construction (e.g., RS codes) to a repair-efficient erasure code construction (e.g., MSR codes) with a configurable sub-packetization level [179].

Prior studies mainly focus on optimizing the repair operations based on RS codes, in which the repair of a chunk can be decomposed into partial linear combinations (§2.1) due to additive associativity. Geometric Partitioning [164] considers regenerating codes and identifies the trade-off between degraded read time and repair efficiency for different chunk sizes. It partitions an object into chunks whose sizes follow a geometric sequence, such that it uses small chunks to reduce the degraded read time and large chunks to achieve high repair efficiency. ParaRC [112] extends OpenEC [113] to parallelize the repair of regenerating codes by balancing the load of retrieving sub-chunks across different nodes, based on the sub-packetization nature (§3.2).

Proactive repair. The above repair algorithms trigger a repair operation upon the detection of a failure. Proactive repair approaches have been proposed to repair soon-to-fail nodes by monitoring the health status of nodes (e.g., by failure prediction algorithms). ToleRAID [59] proposes an adaptive strategy by combining reactive and proactive repair operations for RAID to tolerate tail latencies in SSDs. Hu et al. [63] propose to proactively launch degraded reads to mitigate the access loads of hotspots. FastPR [111] combines the migration of the chunks in a soon-to-fail node and the decoding of the chunks in the soon-to-fail node through erasure coding.

Concurrent repair. Most studies on the repair algorithms for erasure coding focus on single-chunk failures within a stripe. However, *concurrent* multi-chunk failures within a stripe are also possible. Lazy recovery is considered in peer-to-peer networks [15] and data centers [175] to defer the repair operation until a stripe contains a threshold number of failed chunks to significantly reduce the amount of repair bandwidth. Li et al. [107] apply a concurrent repair mechanism to existing regenerating codes and show that the total repair bandwidth is the minimum for most of the cases.

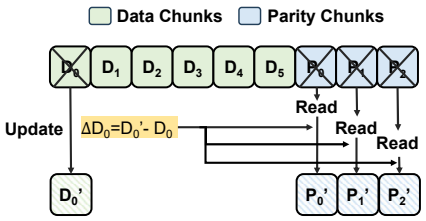


Fig. 12. An in-place update directly updates the data chunk (D_0) and sends the data delta chunk (i.e., ΔD_0) to update the corresponding parity chunks.

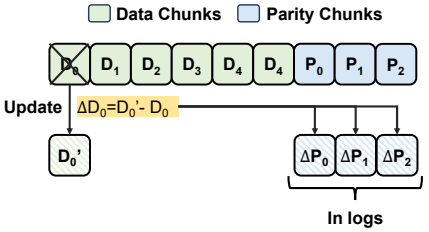


Fig. 13. Parity logging keeps the parity delta chunks (i.e., ΔP_i for $0 \leq i \leq 2$) in logs with the associated parity chunks.

Note that wide-stripe codes (§3.3) are more susceptible to multi-chunk failures for a large stripe size. Liquid cloud storage [121] considers lazy repair for large stripe sizes (in which both n and k are large). It is based on centralized repair, in which a single new node collects k available chunks to decode the lost chunk. HMBR [226] combines centralized repair with independent repair based on repair pipelining [106] to mitigate the repair time for repairing multiple failed chunks in wide stripes.

Cross-rack-aware repair. Several studies consider the repair in rack-based data centers with limited cross-rack bandwidth. CAR [173] computes partially decoded chunks in each rack for RS codes and is shown to minimize the cross-rack repair bandwidth. ClusterSR [172] scatters the reconstructed chunks across racks to minimize and balance the cross-rack upload and download transmissions. Xu et al. [216] propose a data layout based on Pairwise Balanced Design (PBD) to arrange the stripes to achieve a uniform data distribution at both rack and node levels, and further balance the repair loads.

Exploiting the repair capability of erasure coding. While erasure coding incurs high repair overhead, its repair capability can be leveraged to improve the performance of storage systems in different aspects. Li et al. [115] implement erasure coding to resolve the tight tension between the areal density and the read retry rate in hard disks, such that the sector failures inside a hard disk can be repaired by local erasure coding with a latency of tens of hundreds of microseconds, without performing read retries that have a latency of tens or hundreds of milliseconds. Local erasure coding is implemented as a wide stripe with large n and k and small $n-k$, so as to limit the storage overhead.

4.3 Update-Efficient Algorithms

Updates of data chunks in systematic erasure codes trigger updates of parity chunks of the same stripe (§2.3). Earlier studies on erasure-coded storage systems that support decentralized concurrent updates with strong consistency guarantees [8, 42, 46]. Follow-up studies design update-efficient algorithms for erasure-coded storage systems, so as to reduce the number of I/Os in updates and improve the update performance.

In-place updates versus log-based updates. Updates in erasure-coded storage systems can be implemented as *in-place* or *log-based* updates. In-place updates directly overwrite the existing data and parity chunks with the new data and parity chunks, respectively (Figure 12), while log-based updates simply keep the *deltas* (see Equation (1) in §2.3) in an append-only log. Earlier studies propose *parity logging* [80, 176] for RAID by keeping parity deltas (i.e., changes of parity chunks) in an append-only log (Figure 13). Since a parity delta can be calculated by multiplying a data delta

with the corresponding encoding coefficient (Equation (1)), parity logging avoids reading existing parity chunks for updates and hence saves extra I/Os in parity updates. On the other hand, the repair of any failed data chunk needs to retrieve the existing parity chunks and their parity deltas to reconstruct the new parity chunks, thereby triggering extra I/Os.

Several studies propose update-efficient algorithms for distributed storage systems. CodFS [22] extends parity logging for distributed settings, in which in-place updates are applied to data chunks and log-based updates are applied to delta chunks. To save the extra I/Os of reconstructing parity chunks due to parity logging, CodFS reserves a dedicated log next to each parity chunk to keep its parity deltas, so as to reduce the disk I/O overhead in repair operations, the reserved log can be resized based on workload patterns. PARIX [98] reduces the disk I/O overhead in parity updates using speculative partial writes: if a data chunk is updated for the first time, PARIX sends both the old and new data chunks to all parity nodes that store the parity chunks; if the data chunk has been updated before, PARIX directly sends the new data chunks to all the parity nodes and appends it to a dedicated log. Finally, a parity node can read the old and latest data chunks to update a parity chunk. Group-U [140] organizes data chunks into different groups and appoints a relay node for each group to collect data deltas within the same group to calculate the partial parity chunks and update the parity chunks, so as to reduce the update traffic. UCODR [167] focuses on XOR-based erasure codes, and exploits the differences among the parity chunks of the same stripe for efficient parity updates. GOOD [31] temporarily buffers incoming data chunks and later reorganizes them into new stripes, so as to reduce the I/O overhead caused by in-place updates and the performance degradation in future reads caused by log-based updates.

Update routing. Some studies consider efficient routing for transmitting updates in decentralized settings. T-Update [139] distributes the updates across a spanning tree that contains the data nodes that contain the updated data chunks and all corresponding parity nodes that contain the parity chunks of the same stripe. MDTUpdate [239] formulates a cost-based routing optimization problem for multi-block updates in heterogeneous environments.

Data placement. Some studies reduce update traffic with efficient data placement mechanisms based on data access patterns. Parity-Switched Data Placement (PDP) [171] considers XOR-based erasure codes in RAID by issuing updates to the data elements that generate the common parity elements, so as to reduce the number of parity elements being updated. It further designs the generation order of the parity elements. CASO [168] further identifies the data chunks that are accessed within a short time and places such correlated data chunks in the same stripe.

Cross-rack-aware updates. Some studies consider updates in erasure-coded storage systems in rack-based data centers with limited cross-rack bandwidth. CAU [169] reduces cross-rack parity updates by selectively transmitting data deltas or parity deltas across racks based on the number of updated data chunks in a stripe. RackCU [45] extends CAU with an optimal algorithm that minimizes cross-rack updates.

4.4 Consensus Algorithms for Erasure Coding

Consensus algorithms, such as Paxos [94, 95] and Raft [132], are vital for distributed storage systems to provide reliable services in the face of accidental failures. They record operation commands as log entries and replicate the entries across multiple nodes, such that the state machines of different nodes can execute the recorded commands in the right order and obtain the correct results, even in the presence of node failures. The consensus algorithms are prone to introduce high storage and network overhead: to tolerate f node failures (where $f > 0$), the consensus algorithms have to replicate log entries across $2f + 1$ nodes. Simply using replication in consensus algorithms to

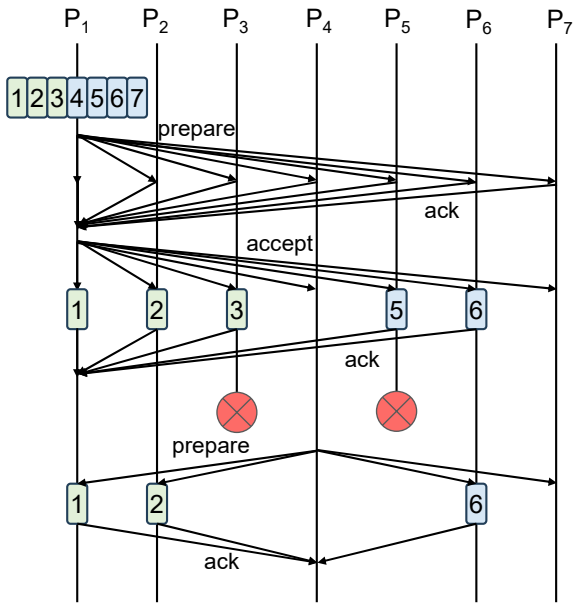


Fig. 14. Example of RS-Paxos [127], which encodes data with $(n, k) = (7, 3)$ RS codes and can tolerate any two node failures.

promise consistency will further deteriorate the system performance, since the data volume stored and operated in large-scale storage systems is fast growing.

RS-Paxos [127] leverages erasure coding to reduce the network and storage overhead in consensus protocols. It trades liveness (i.e., the number of tolerable node failures) for reduced network and storage consumptions, where a storage system with $n = (2f + 1)$ nodes under RS-Paxos is proven to tolerate $f - \frac{k-1}{2}$ node failures, which is smaller than f when $k > 1$. Figure 14 shows an example of RS-Paxos, which encodes data with RS codes under $(n, k) = (7, 3)$ and tolerates any two node failures out of the seven nodes. In this example, node P_1 sends a prepare message to seven nodes, which acknowledge the receipts. Then, P_1 sends an accept request that encodes a message with the $(7, 3)$ RS code and waits for at least five acknowledgments. If another node P_4 requests the encoded message, it retrieves the encoded chunks from at least three nodes to decode the message. Here, the message can be decoded even with the loss of two accept requests and two encoded chunks.

Several follow-up studies examine consensus in erasure coding. Pando [185] applies erasure coding to geo-distributed storage and reduces the wide-area latencies in reads and writes while maintaining consensus. To sustain the liveness as in Raft [132] and Paxos [94, 95], CRaft [193] suggests using two methods to maintain the log entries: (i) if the leader can contact at least $f + k$ healthy nodes, it prefers encoding the log entries; and (ii) if the number of healthy nodes is larger than f but smaller than $f + k$, CRaft chooses to store the replications of the log entries. CRaft is proven to attain the same liveness level as Raft, but it still introduces considerable network and bandwidth overhead. HRaft [77] still employs erasure coding when the number of failed nodes is larger than $n - f - k$ by adjusting the placement of coded chunks and replicating some coded chunks to healthy nodes (rather than using replication). While the above studies consider the optimization of storage and network overhead under fixed coding configurations, FlexRaft [233]

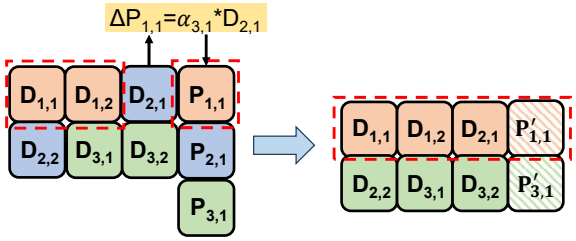


Fig. 15. Example of ERS [204], which converts the $(3, 2)$ RS-coded stripes into $(4, 3)$ RS-coded stripes. It eliminates data relocation. It only calculates the parity delta chunk based on the newly added data chunks (e.g., $D_{2,1}$) and updates the corresponding parity chunk (e.g., $P_{1,1}$).

further minimizes the storage and network overhead by dynamically exploring the best-fit coding configurations based on the number of failed nodes.

4.5 Redundancy Transitioning for Erasure Coding

The configuration of erasure coding parameters exhibits a trade-off across access performance, storage efficiency, and reliability. Conventional deployments use a single erasure code with fixed coding parameters for all data [1, 41, 129]. However, such a ''one-size-fits-all'' approach lacks efficiency in adaptation to the varying access characteristics and reliability requirements. In the face of increasing storage demands, modern storage systems support *redundancy transitioning* for erasure-coded data to balance the trade-off between access performance and storage efficiency. By redundancy transitioning, we refer to adjusting the erasure coding parameters and re-encoding the existing coded data with new coding parameters. Redundancy transitioning has been widely explored for its advantages in adaptation to workload dynamics [210], disk reliability changes [82--84] and generation of wide stripes [61, 85]. Albeit the benefits, redundancy transitioning often incurs substantial network traffic and I/O overheads in a distributed setting, especially in re-encoding the existing data and re-distributing the newly coded data. We now overview recent studies that address redundancy transitioning.

From replication to erasure coding. Earlier studies address redundancy transitioning from replication to erasure coding. AutoRAID [196] combines replication and RAID in a single disk-array controller. It leverages access characteristics to perform redundancy transitioning between replication (for hot data) to RAID (for cold data). DiskReduce [39] and EAR [104] address the transitioning from replication to erasure coding in HDFS [3]. DiskReduce performs asynchronous encoding, where data are initially replicated when being stored, and later encoded with erasure coding in the background. EAR designs data placement algorithms for replicas to mitigate the traffic incurred by transitioning from replication to erasure coding. RapidRAID [136] proposes a pipelined encoding approach for replicated data to speed up the re-encoding operations and reduce the traffic for redundancy transitioning.

Changing the erasure coding parameters. Some studies address the redundancy transitioning between erasure codes with different coding parameters. HACFS [210] supports efficient transitioning between two erasure codes in HDFS, where the hot data is encoded with high-redundancy codes for access efficiency, and the cold data is encoded with low-redundancy codes for storage savings. Zebra [197] leverages Cauchy-based RS codes to mitigate the traffic for redundancy transitioning. SRS [180] and ERS [204] co-design the encoding matrix and data placements for RS codes to mitigate the I/O overheads in redundancy transitioning. Figure 15 shows an example of ERS, which converts

the (3,2) RS-coded stripes into (4,3) RS-coded stripes. In this example, ERS does not need to perform data relocation. It only needs to transmit a parity delta chunk for parity chunk updates.

Scaling is a class of redundancy transitioning for erasure coding. It not only renews the erasure coding parameters, but also enforces an even storage layout across storage nodes; in some cases, it targets a uniform distribution of data and parity chunks. Earlier studies design efficient scaling approaches for RAID, for example, in RAID-0 [229, 237], RAID-5 [198, 230, 231] and RAID-6 [199, 200, 228], respectively. However, these approaches are tailor-made for RAID and cannot tolerate more than two failures. Recent studies consider scaling for general erasure coding. ScaleRS [69] proposes efficient scaling schemes for RS codes to mitigate the overhead of data migration and parity updates. Wu et al. [206] design efficient encoding matrices for Cauchy-RS codes to mitigate the scaling I/O overhead. NCScale [64] leverages network coding to minimize the scaling bandwidth. Wu et al. [205] study the trade-off between repair and scaling performance for locally repairable codes in hierarchical storage and propose data placements that achieve the optimal trade-off.

Code conversion is another class of redundancy transitioning for erasure coding. Code conversion simplifies the redundancy transitioning for practicality, such that it does not require data to be evenly distributed after transitioning (which is required for scaling). Maturana et al. [124] initiate the study of convertible codes, with a focus on minimizing the conversion I/Os. Follow-up studies on code conversion focus on minimizing the conversion bandwidth [125]. Some studies address a special case of code conversion called *stripe merging*, where multiple short stripes (i.e., with a small (n, k)) are merged into a wide stripe (i.e., with a large (n, k)). StripeMerge [223] proposes efficient heuristics for stripe merging in RS codes to mitigate transitioning bandwidth. Wu et al. [203] design optimal data placements for stripe merging in locally repairable codes to minimize cross-rack transitioning bandwidth under a hierarchical setting.

Disk-adaptive redundancy. Recent studies identify the heterogeneity of disk failure rates in production storage systems, and propose *disk-adaptive redundancy* to balance the trade-off between storage overhead and reliability. HeART [84] suggests appropriate coding parameters for redundancy transitioning to meet both reliability requirements and target storage savings based on predicted disk failure rates. PACEMAKER [83] realizes efficient redundancy transitioning while providing storage savings and fault tolerance in distributed storage. It groups disks with similar disk failure rates, and proactively performs redundancy transitioning between different disk groups based on the changes of disk failure rates. Tiger [82] further improves the storage efficiency and reliability over PACEMAKER. It relaxes the data placement constraints of PACEMAKER, such that each disk group can accommodate disks with different disk failure rates for higher risk diversity.

4.6 Reliability Evaluation

Since erasure coding incurs a high repair penalty, several studies examine its impact on reliability. We review related studies on the reliability evaluation of erasure-coded storage from two perspectives, namely modeling and simulation.

Modeling. Weatherspoon and Kubiatowicz [194] quantitatively compare the reliability of replication and erasure coding in terms of the mean-time-to-failure (MTTF), and show that erasure coding has a significantly higher MTTF than replication under the same storage overhead as well as significantly less storage overhead than replication under the same reliability. Rodrigues and Liskov [160] compare the reliability of replication and erasure coding in peer-to-peer distributed hash tables (DHTs), and show that the benefits of erasure coding may be limited and even negated by the complexity of deploying erasure coding.

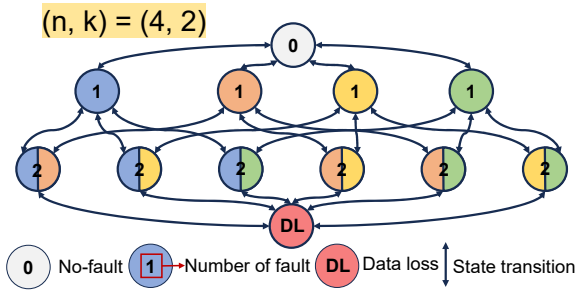


Fig. 16. Example of a reliability analysis based on Markov modeling for a $(n, k) = (4, 2)$ RS code. The data will be lost once the number of failures exceeds three. The circles mean the system state with different numbers of failures.

Markov modeling is extensively used to characterize the reliability of erasure-coded storage systems in terms of the MTTF (or mean-time-to-data-loss (MTTDL)), under the assumption that the failure and repair times are independently and exponentially distributed with constant rates. Figure 16 shows an example of reliability analysis under Markov modeling. Thomasian et al. [183] model the reliability of RAID, with the consideration of sector errors and disk scrubbing. Ford et al. [41] model the reliability of erasure coding in Google storage clusters at the granularity of the stripe and show that erasure coding has higher reliability than replication. Rao et al. [150] model node and disk failures and show that repair rates significantly impact reliability. Iliadis et al. [72] model the reliability of disk scrubbing and intra-disk redundancy schemes. Markov modeling has also been used to motivate the design of erasure code constructions [61, 62, 66, 162]. Iliadis et al. [73] propose non-Markov modeling with enhanced equations for MTTDL calculations. Later, Iliadis [71] models latent errors and device failures and shows that latent errors cause MTTDL degradations.

Simulation. Markov modeling takes some assumptions, and its correctness is questioned about its usefulness in reliability evaluation [50]. Greenan proposes the *High Fidelity Reliability Simulator (HFRS)* for reliability simulation on disk arrays [47], which performs Monte Carlo discrete event simulation. It reports *NOrmalized Magnitude of Data Loss (NOMDL)*, which evaluates the expected amount of data loss (in bytes) within a specified mission time.

Xin et al. [212] study the reliability of large-scale storage systems via simulation under various factors that influence the system reliability, and propose a distributed recovery algorithm that can reduce the recovery time to improve the system reliability. Greenan et al. [49] study the impact of the placement of erasure-coded symbols for XOR-based erasure codes via simulation, and propose algorithms that can maximize reliability. Hall *et al.* [57] and Zhang et al. [232] also study the effects of data placement in data centers by simulation. The difference is that Zhang et al. [232] consider a hierarchical data center, and show that a hierarchical block placement, which places multiple blocks of a stripe in the same rack, contributes to higher reliability. Epstein *et al.* [36] study the cumulative effect of simultaneous failures on the repair time. Han et al. [58] study how correlated failures influence the reliability of SSD-based data centers. Wang et al. [190] study the design of Multi-level erasure coding (MLEC) via simulation, and show that MLEC provides high reliability compared with state-of-the-art erasure codes (e.g., locally repairable codes).

5 ERASURE CODING FOR EMERGING ARCHITECTURES

In this section, we study the deployment and design considerations of erasure coding in emerging architectures.

5.1 Erasure Coding for Flash Storage

Flash-based solid-state drives (SSDs) have been increasingly used in modern data centers as they provide higher speeds, reliability, storage density, and power efficiency than traditional traditional hard disk drives (HDDs) [147]. Unlike HDDs, SSDs have different access characteristics and error patterns. Specifically, flash is a write-once medium, and flash cells can be re-written only after an erasure operation. However, flash cells can only endure a limited number of erasures depending on the number of bits in each cell, and will become unusable when the erasure limit is reached. Conventional erasure coding may be inefficient for flash memory as it does not consider the erasure limits of flash memory.

Error correction codes are commonly used in SSDs to provide reliability and endurance guarantees. Low-density parity-check (LDPC) codes have been widely used in SSDs due to their low decoding complexity and high error correction capabilities. Zhao et al. [236] propose different techniques to improve LDPC decoding performance. To improve endurance, several studies examine *write-once memory* (WOM) codes, which support the in-place updates of the stored bits in one direction (e.g., from '0' to '1') without erasing the stored data. However, directly deploying WOM codes in flash-based SSDs will increase storage overhead and require an additional read before each additional write [219]. To reduce the extra storage overhead caused by WOM codes, Yadgar et al. [219] propose a Reusable SSD, which reuses invalid and over-provisioned space for additional writes. Margalia et al. [123] further implement WOM codes in an OpenSSD board with four multi-level cell (MLC) flash chips to study the impact on cell wearing and energy consumption introduced by the deployment of WOM codes. They also study the end-to-end flash translation layer (FTL) implementation. They reveal that the page reuse of MLC flash cells can utilize only half of the pages, WOM codes can save energy only when the write operations are reduced, and the reduction of erasure operations does not necessarily increase the SSD endurance. Jaffer et al. [74, 75] treat the bits stored in quad-level cell (QLC) flash chips as monotonically increased voltage and devise a new family of WOM codes that leverages all intermediate voltage levels to improve the writable capacity before performing erasure.

5.2 Erasure Coding for In-Memory Storage

In addition to providing data reliability for persistent storage, erasure coding is also extensively employed in in-memory storage to provide memory-efficient fault tolerance. Deploying erasure coding in in-memory storage usually faces the following design considerations. First, object sizes may vary across workloads, while small objects increase the metadata overhead for erasure-coded stripe management. Thus, the erasure coding strategies for in-memory storage should address different deployment issues, including the encoding granularity (e.g., chunk-level or object-level), the fault-tolerance policy (e.g., using the combination of replication and erasure coding, or solely using erasure coding), and the update approach (e.g., in-place updates or out-of-place updates). Second, accesses to objects are often highly skewed and some objects may receive more accesses than others. The access skewness imposes new requirements to consider both memory efficiency and access performance when deploying erasure coding in in-memory storage, such that erasure coding should save memory footprints while maintaining high access performance.

Coding policy. Since memory resources are stringent, applying replication to objects for fault tolerance significantly increases storage overhead. An earlier work LH$^*_{RS}$ [117] proposes a high-availability scalable distributed data structure based on erasure coding and organizes key-value items in buckets. Rashmi et al. [151] show that selective replication for popular objects cannot effectively achieve load balancing. They propose EC-Cache [151], which encodes an object into n coded chunks and distribute the chunks across nodes, so as to spread the load of read requests for higher I/O performance. It also employs late binding to retrieve $k+\Delta$ chunks (where $1 \leq \Delta \leq n-k$) chunks and uses the first arrived k chunks to restore the requested object. Shankar et al. [166] conduct a systematic analysis to study the trade-off of replication and erasure coding for key-value stores on modern high-performance computing clusters. They also compare different online erasure coding designs that distribute encoding and decoding functions to clients and servers. Since objects have small and scattered metadata (e.g., hash tables and allocation information), Cocytus [23] is a key-value store that employs erasure coding for large application data (e.g., values) and uses primary replication for small-sized and scattered data (e.g., metadata and keys). Prior field reports have shown that small objects dominate the storage workloads [222, 225]. To save storage overhead with high reliability guarantees, MemEC [225] aggregates multiple small objects into a data chunk and applies cuckoo hashing [134] to improve the storage efficiency of indexes. CoREC [34] couples data resilience techniques with data access patterns. It identifies write-hot and write-cold data, and replicates the write-hot data while applying erasure coding to write-cold data so as to balance the storage overhead and parity update costs. InfiniCache [187] applies erasure coding to server cloud functions so as to tolerate data loss caused by function reclamation and mitigate tail latencies caused by straggling functions. It realizes encoding and decoding functionalities in the client library to improve the performance of the chunk streaming pipeline.

Update efficiency. Since the objects in memory may be frequently updated, some studies improve the update efficiency in erasure-coded in-memory storage. Xia et al. [208] find that a stripe may comprise multiple small objects. They propose a grouping-update mechanism, which buffers multiple small updates in a time window and organizes them into multiple independent groups, ensuring that the small updates of the same group can be performed in parallel, so as to reduce parity update traffic. BCStore [110] employs out-of-place updates to organize the incoming updates in a new stripe so as to batch multiple in-place updates. It also moves the valid chunks from the stripes with the most stale chunks to the ones with the fewest stale chunks to reclaim memory space. F-Write [214] identifies that multiple round-trips and encoding operations become the new performance bottleneck of update operations in RDMA-supported in-memory clusters. It uses the WRITE verbs to directly modify remote transaction logs to reduce network round-trips. It also records transaction entities of multiple writes in the log at the parity nodes, which will be submitted speculatively to reduce encoding latency. To balance the update performance and memory costs, LogECMem [25] places one parity chunk to perform in-place updates in DRAM for fast repair with low memory overhead, while keeping other $n-k-1$ parity chunks on disks with parity logging for efficient updates.

Object redistributions. Some studies consider object redistributions for in-memory storage, which may be caused by caching policy, redundancy transitioning, and storage scaling. To balance the trade-off between the number of chunks being cached and the access latency, Agar [56] adjusts the cache policy based on the live information of data popularity and the access latencies to different data storage sites. SRS [180] and ERS [204] reduce I/O operations in redundancy transitioning by maintaining key-to-node mapping and using enlarged encoding matrices (§4.5). TEA [213] considers the transitioning from replication to erasure coding for in-memory storage. It gathers a replica of each chunk in a dedicated rack and encodes them within the same rack, so as to reduce

the migration of data chunks across racks during encoding. By considering the access popularity of chunks, PaRS [240] adaptively changes the replication degree of data chunks in stripes to tune the access parallelism. ECHash [26] maps the sub-chunks into multiple nodes and organizes nodes into multiple isolated zones, thereby reducing the overhead of object redistribution and parity updates under storage scaling.

5.3 Erasure Coding for Disaggregated Memory

Traditional monolithic servers tightly couple memory with specific processors and need to scale all computing and memory resources simultaneously, thereby leading to resource underutilization (e.g., around 40-60% of memory utilization [51, 165]). Disaggregated memory decouples computing and memory resources into independent and failure-isolated computing and memory pools, respectively, so as to improve elasticity and scalability. The compute pool comprises multiple CPU blades and a small amount of memory space (e.g., 1-10 GB [192]) for application data caching, while the memory pool has multiple memory blades (e.g., DRAM DIMMs and persistent memory DIMMs) and small computation power only for memory allocations and network interconnections [188]. Disaggregated memory also challenges the deployment of erasure coding, as its high available network bandwidth makes encoding and decoding latencies become non-negligible. This violates the assumptions made in prior erasure coding studies (e.g., regenerating codes [32] and repair parallelization techniques (e.g., [68, 106, 126])) that commonly treat the network and storage bandwidth as the performance bottleneck. Second, disaggregated memory applications usually allocate variable-sized objects, while erasure coding usually operates in fixed-sized chunks. This difference complicates the swap mechanism in memory pooling.

MicroEC [103] shows that the encoding latency is comparable with the RDMA latency in disaggregated memory and proposes to coordinate data encoding and RDMA transmission to realize efficient pipelining. Hydra [96] applies erasure coding in disaggregated cluster memory with a latency of single-digit microseconds by dividing each 4-KB page into smaller pieces and applying asynchronously encoded writes to hide encoding latency. However, this approach may result in multiple network communications to fetch a single page. Carbink [242] exposes far memory and performs the swapping strategy at the granularity of *spans* that comprise multiple memory pages containing objects with similar sizes. It allows a compute node to fetch a memory region using a single network request.

5.4 Erasure Coding for Programmable Switches

Update and repair operations in erasure coding require multiple nodes to transmit data cooperatively, thereby introducing substantial network communications. The emergence of programmable switches has opened up new opportunities to reduce the network round-trips across nodes, as they have computation power and memory space for performing in-network computing when data is transmitted on the fly. However, programmable switches have limited resources for data computations, thereby challenging the deployment of erasure coding. First, the available on-chip memory in programmable switches is scarce, while erasure coding needs to retrieve and buffer multiple surviving chunks for repair operations. The memory consumption also increases significantly if multiple chunks are transferred in different rates. Second, programmable switches only support simple integer arithmetic operations (e.g., addition and subtraction), while erasure coding is realized by Galois Field arithmetic (§2.1). Thus, offloading encoding and decoding operations to programmable switches is a non-trivial matter.

Xia et al. [207] and XORInc [189] propose to aggregate requested data in switches to reduce the network transmission time. However, both of them simply use simulation for performance evaluation. NetEC [148] offloads erasure coding operations to programmable switching ASICs,

which realize Galois Field vector dot-product operations using table lookups and partial XOR-sum updates. Jiang et al. [79] offload matrix operations of random linear network coding to programmable switches for acceleration. INC-EC [78] employs programmable switches to perform data encoding and decoding operations, in which multiplications in the Galois Field are realized by table lookups and additions are realized by the switch arithmetic logic unit. Paint [209] is a parallelized in-network aggregation framework for fast failure recovery by implementing multiple parallelized repair pipelines in programmable switches.

5.5 Erasure Coding for Edge-Cloud Storage

Storage tiering is a new storage paradigm that separates data storage into different tiers according to data size and access popularity. Edge-cloud storage is a representative application of storage tiering, where the edge layer caches frequently accessed data close to clients for low access latency, and the cloud layer provides a large volume of storage space for data persistence. Edge-cloud storage has some unique characteristics; for example, edge nodes have limited storage and computation resources and often need to serve latency-sensitive requests. This brings new design considerations and challenges to the deployment of erasure coding, such as how to design cache policies for load balancing in edge nodes, how to reduce the synchronization traffic between the edge and cloud layers, and how to distribute the cached chunks across edge nodes to mitigate access latencies.

Several studies address the aforementioned challenges. Konwar et al. [88] use regenerating codes in the cloud layer to reduce read communications when needing to recreate the objects (in the edge layer) using the coded chunks (from the cloud layer). Yang et al. [221] find that large CDNs cannot address the cache miss ratio spike and write imbalance in a memory-efficient way. They propose C2DN that employs erasure coding in the CDN architecture and uses parity chunks to re-balance the write load across edge nodes. Jin et al. [81] incorporate proximity, encoding, and transmission constraints in formulating the erasure-coding-based edge data placement problem, which aims to minimize the storage overhead while providing fast service for all users. Liu et al. [118] design an adaptive and scalable caching strategy to decide the data chunks to be cached based on the data popularity and network latency. ELECT [159] employs a hybrid redundancy approach that replicates a limited amount of hot KV pairs in the hot tier (e.g., the edge layer) and encodes large amounts of cold KV pairs in the cold tier (e.g., the cloud layer) via erasure coding to achieve high access performance and storage saving. It also provides a redundancy transitioning approach to convert replication to erasure coding across the tiers.

6 OPEN PROBLEMS AND FUTURE DIRECTIONS

While erasure coding for storage systems has been extensively studied for decades, there still are several open problems and future directions that have research potential.

Erasure coding for DNA storage. DNA storage is a new storage method that represents data information in the form of DNA sequences, which can store large amounts of information in a less expensive, more energy-efficient, and more durable manner. The rationale of DNA storage is to convert digital data into four DNA bases: adenine (A), cytosine (C), guanine (G), and thymine (T). It is estimated that DNA storage can squeeze all the information generated by humanity (which is around 33 zettabytes of data by 2025) into a ping-pong ball [2]. However, DNA storage still has a major challenge that errors may arise in the synthesizing (i.e., generating a pool of oligonucleotide strands) and sequencing (i.e., recovering the digital message) processes. There are three possible errors associated with DNA storage: substitutions of one base by another, deletion errors of nucleotides, and insertion errors of nucleotides. The errors in DNA storage have two patterns. First, it is reported that insertions and deletions are two predominant errors in DNA

storage, which occupy 6.2% and 5.7% of error probabilities, respectively [11]. Second, DNA storage usually encounters errors during synthesis and sequencing, where most of the errors arise from the sequencing process again [21]. It is vital to design new families of erasure codes and decoding techniques based on the error frequency and error occurrence for DNA storage.

AI and erasure coding. There are two directions when trying to couple AI and erasure coding together. The first is AI for erasure coding, which employs AI techniques to explore the code constructions and the deployment for erasure coding. Specifically, traditional error correction codes are constructed based on coding theory, such as using generator matrices to construct linear erasure codes and constructing parity-check matrices to define LDPC codes. Recently, some studies [35, 70] leverage AI techniques (e.g., recurrent neural network and reinforcement learning) to construct error correction codes in addition to the expert knowledge in coding theory. Thus, it is possible to apply AI techniques to predict the best coding parameters for erasure codes. Also, AI techniques can analyze the data access pattern (including access correlation and hotness), so as to guide the deployment of erasure coding in storage systems, such as erasure code constructions to reduce parity updates, erasure-coded stripe management to improve the cache hit ratio, and the distribution of erasure-coded stripes to improve the access throughput.

Existing erasure coding designs are *reactive* to system changes, meaning that they take actions only after the occurrence of system events, such as repairing lost data chunks after data loss or relocating data chunks after identifying hotspots. AI techniques can analyze system logs to predict the reliability of hardware (e.g., memory DIMMs [27, 128], disks [120, 218], and switches [234]) and learn the reliability of storage systems. Once AI techniques predict that the system reliability degrades, they can immediately notify system administrators to *proactively* launch actions, such as redundancy transitioning (i.e., adding more redundancy for improving reliability), frequent inspections (i.e., reporting potential failures), and data relocation (i.e., mitigating the impact of unexpected data loss). In addition, AI techniques can predict the change of access patterns, which is beneficial for the cache policies in edge-cloud storage as well as the prefetching strategies in programmable switches and disaggregated memory systems. We believe that using AI techniques for better erasure coding management is a promising direction.

The second is erasure coding for AI. With the increase of computational complexity in machine learning models (e.g., generative language models), model training becomes expensive and long-lasting (e.g., taking hundreds of thousands of compute days [4]). Thus, how to maintain the robustness of model training is critical. Erasure coding is considered a promising solution for large model training [89], since it provides memory-efficient resilience for computations and storage. However, it still faces several open issues, such as how to perform periodical encoding without compromising model training performance, how to choose or design appropriate codes for the non-linear components in machine learning models, and how to coordinate the memory usage between erasure coding and AI computations.

7 SUMMARY AND CONCLUSIONS

Erasure coding is a widely used approach to guarantee data reliability in a storage-efficient manner, and the research on erasure coding has been evolving with the development of storage systems. In this survey, we conduct an in-depth study on erasure coding from a systems perspective. Specifically, we classify the existing studies on erasure coding for storage systems into three categories: (i) new constructions for erasure codes, in which we review the families of erasure codes that improve traditional RS codes in terms of encoding and decoding performance (e.g., XOR-based erasure codes), repair performance (e.g., regenerating codes, locally repairable codes, and piggybacking codes), and storage efficiency (e.g., sector-disk codes); (ii) algorithmic designs for erasure coding,

in which we review the optimization techniques for erasure-coded storage systems in different aspects, including encoding/decoding, repair, update, consensus, redundancy transitioning, and reliability evaluation; and (iii) erasure coding for emerging architectures, in which we describe the deployment of erasure coding under different architectures. We finally pose the open problems and future directions that may inspire further research on erasure coding.

REFERENCES

[1] Accessed in March 2024. Backblaze Vaults: Zettabyte-Scale Cloud Storage Architecture. https://www.backblaze.com/blog/vault-cloud-storage-architecture/.

[2] Accessed in March 2024. DNA: The Ultimate Data-Storage Solution. https://www.scientificamerican.com/article/dna-the-ultimate-data-storage-solution/.

[3] Accessed in March 2024. HDFS erasure coding. https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSErasureCoding.html.

[4] Accessed in March 2024. Meeting the Operational Challenges of Training LLMs. https://thenewstack.io/meeting-the-operational-challenges-of-training-llms/.

[5] Accessed in March 2024. Tencent ultra-cold storage system optimization with Intel ISA-L - a case study. https://www.intel.cn/content/dam/develop/external/us/en/documents/case-study-of-tencent-ultra-cold-storage-system-optimization-720747.pdf.

[6] Accessed in March 2024. Yahoo Cloud Object Store - Object Storage at Exabyte Scale. https://yahooeng.tumblr.com/post/116391291701/yahoo-cloud-object-store-object-storage-at.

[7] Michael Abd-El-Malek, William V. Courtright II, Chuck Cranor, Gregory R. Ganger, James Hendricks, Andrew J. Klosterman, Michael Mesnier, Manish Prasad, Brandon Salmon, Raja R. Sambasivan, Shafeeq Sinnamohideen, John D. Strunk, Eno Thereska, Matthew Wachs, and Jay J. Wylie. 2005. Ursa Minor: versatile cluster-based storage. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST'05)*. 59--72.

[8] Marcos Kawazoe Aguilera, Ramaprabhu Janakiraman, and Lihao Xu. 2005. Using erasure codes efficiently for storage in a distributed system. In *Proceeding of the 2005 International Conference on Dependable Systems and Networks (DSN '05)*. 336--345.

[9] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. 2000. Network Information Flow. *IEEE Transactions on Information Theory* 46, 4 (Jul 2000), 1204--1216.

[10] Fabien André, Anne-Marie Kermarrec, Erwan Le Merrer, Nicolas Le Scouarnec, Gilles Straub, and Alexandre van Kempen. 2014. Archiving Cold Data in Warehouses with Clustered Network Coding. In *Proceedings of the 9th European Conference on Computer Systems (EuroSys'14)*. 21:1--21:14.

[11] Philipp L Antkowiak, Jory Lietard, Mohammad Zalbagi Darestani, Mark M Somoza, Wendelin J Stark, Reinhard Heckel, and Robert N Grass. 2020. Low cost DNA data storage using photolithographic synthesis and advanced information reconstruction and error correction. *Nature communications* 11, 1 (2020), 5345.

[12] Yunren Bai, Zihan Xu, Haixia Wang, and Dongsheng Wang. 2019. Fast recovery techniques for erasure-coded clusters in non-uniform traffic network. In *Proceedings of the 48th International Conference on Parallel Processing (ICPP '19)*. 1--10.

[13] S. B. Balaji, M. Nikhil Krishnan, Myna Vajha, Vinayak Ramkumar, Birenjith Sasidharan, and P. Vijay Kumar. 2018. Erasure coding for distributed storage: an overview. *Science China Information Sciences* 61, 100301 (2018), 100301:1--100301:45.

[14] Shobana Balakrishnan, Richard Black, Austin Donnelly, Paul England, Adam Glass, Dave Harper, Sergey Legtchenko, Aaron Ogus, Eric Peterson, and Antony Rowstron. 2014. Pelican: A building block for exascale cold data storage. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI '14)*. 351--365.

[15] Ranjita Bhagwan, Kiran Tati, Yuchung Cheng, Stefan Savage, and Geoffrey M Voelker. 2004. Total Recall: System support for automated availability management. In *Proceedings of the 1st Symposium on Networked Systems Design and Implementation (NSDI '04)*. 337--350.

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

[17] Mario Blaum, Jim Brady, Jehoshua Bruck, Jai Menon, and Alexander Vardy. 2002. *The EVENODD Code and its Generalization: An Efficient Scheme for Tolerating Multiple Disk Failures in RAID Architectures.* Wiley-IEEE, Chapter 8, 93--117.

[18] Mario Blaum, James Lee Hafner, and Steven Hetzler. 2013. Partial-MDS codes and their application to RAID type of architectures. *IEEE Transactions on Information Theory* 59, 7 (2013), 4510--4519.

[19] Johannes Blömer, Malik Kalfane, Richard Karp, Marek Karpinski, Michael Luby, and David Zuckerman. 1995. *An XOR-Based Erasure-Resilient Coding Scheme.* Technical Report ICSI TR-95-048. International Computer Sciences

Institute.
[20] Eric Burgener and John Rydning. 2022. High Data Growth and Modern Applications Drive New Storage Requirements in Digitally Transformed Enterprises. https://www.delltechnologies.com/asset/en-my/products/storage/industry-market/h19267-wp-idc-storage-reqs-digital-enterprise.pdf. International Data Corporation Doc. #US49359722.
[21] Ben Cao, Xiaokang Zhang, Shuang Cui, and Qiang Zhang. 2022. Adaptive coding for DNA storage with high storage density and low coverage. *NPJ systems biology and applications* 8, 1 (2022), 23.
[22] Jeremy C. W. Chan, Qian Ding, Patrick P. C. Lee, and Helen H. W. Chan. 2014. Parity logging with reserved space: Towards efficient updates and recovery in erasure-coded clustered storage. In *Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST '14)*. 163--176.
[23] Haibo Chen, Heng Zhang, Mingkai Dong, Zhaoguo Wang, Yubin Xia, Haibing Guan, and Binyu Zang. 2017. Efficient and available in-memory KV-store with hybrid erasure coding and replication. *ACM Transactions on Storage* 13, 3 (2017), 167--180.
[24] Henry C. H. Chen, Yuchong Hu, Patrick P. C. Lee, and Yang Tang. 2014. NCCloud: A Network-Coding-Based Storage System in a Cloud-of-Clouds. *IEEE Trans. Comput.* 63, 1 (Jan 2014), 31--44.
[25] Liangfeng Cheng, Yuchong Hu, Zhaokang Ke, Jia Xu, Qiaori Yao, Dan Feng, Weichun Wang, and Wei Chen. 2021. LogECMem: Coupling erasure-coded in-memory key-value stores with parity logging. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*. 89--103.
[26] Liangfeng Cheng, Yuchong Hu, and Patrick P. C. Lee. 2019. Coupling decentralized key-value stores with erasure coding. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC '19)*. 377--389.
[27] Zhinan Cheng, Shujie Han, Patrick PC Lee, Xin Li, Jiongzhou Liu, and Zhan Li. 2022. An in-depth correlative study between DRAM errors and server failures in production data centers. In *Proceedings of the 41st International Symposium on Reliable Distributed Systems (SRDS'22)*. 262--272.
[28] Peter Corbett, Bob English, Atul Goel, Tomislav Grcanac, Steven Kleiman, James Leong, and Sunitha Sankar. 2004. Row-diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST'04)*. 1--14.
[29] Frank Dabek, Jinyang Li, Emil Sit, James Robertson, M. Frans Kaashoek, and Robert Morris. 2004. Designing a DHT for low latency and high throughput. In *Proceedings of the 1st USENIX Symposium on Networked Systems Design and Implementation (NSDI'04)*. 649--667.
[30] Anwitaman Datta and Frédérique Oggier. 2013. An Overview of Codes Tailor-made for Better Repairability in Networked Distributed Storage Systems. *ACM SIGACT News* 44, 1 (2013), 89--105.
[31] Haiwei Deng, Ranhao Jia, and Chentao Wu. 2021. A graph-assisted out-of-place update scheme for erasure coded storage systems. In *Proceedings of the 50th International Conference on Parallel Processing (ICPP '21)*. 1--10.
[32] Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin J. Wainwright, and Kannan Ramchandran. 2010. Network coding for distributed storage systems. *IEEE Transactions on Information Theory* 56, 9 (2010), 4539--4551.
[33] Alexandros G Dimakis, Kannan Ramchandran, Yunnan Wu, and Changho Suh. 2011. A Survey on Network Codes for Distributed Storage. *Proc. IEEE* (2011), 476--489.
[34] Shaohua Duan, Pradeep Subedi, Philip Davis, Keita Teranishi, Hemanth Kolla, Marc Gamell, and Manish Parashar. 2020. CoREC: Scalable and resilient in-memory data staging for in-situ workflows. *ACM Transactions on Parallel Computing* 7, 2 (2020), 12:1--12:29.
[35] Ahmed Elkelesh, Moustafa Ebada, Sebastian Cammerer, and Stephan Ten Brink. 2019. Decoder-tailored polar code design using the genetic algorithm. *IEEE Transactions on Communications* 67, 7 (2019), 4521--4534.
[36] Amir Epstein, Elliot K Kolodner, and Dmitry Sotnikov. 2016. Network aware reliability analysis for distributed storage systems. In *Proceedings of the IEEE 35th Symposium on Reliable Distributed Systems (SRDS '16)*. 249--258.
[37] Kyumars Sheykh Esmaili, Lluis Pamies-Juarez, and Anwitaman Datta. 2013. CORE: Cross-Object Redundancy for Efficient Data Repair in Storage Systems. In *Proceedings of 2013 IEEE International Conference on Big Data (BigData'13)*. 246--254.
[38] Vero Estrada-Galinanes, Ethan Miller, Pascal Felber, and Jehan-François Pâris. 2018. Alpha entanglement codes: Practical erasure codes to archive data in unreliable environments. In *Proceedings of the 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '18)*. 183--194.
[39] Bin Fan, Wittawat Tantisiriroj, Lin Xiao, and Garth Gibson. 2009. DiskReduce: RAID for data-intensive scalable computing. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage (PDSW '09)*. 6--10.
[40] Andrew Fikes. 2010. Storage architecture and challenges. http://cloud.google.com/files/storage_architecture_and_challenges.pdf.
[41] Daniel Ford, François Labelle, Florentina I Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. 2010. Availability in globally distributed storage systems. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI'10)*. 61--74.

[42] Svend Frolund, Arif Merchant, Yasushi Saito, Susan Spence, and Alistair Veitch. 2004. A decentralized algorithm for erasure-coded virtual disks. In *Proceeding of the 2004 International Conference on Dependable Systems and Networks (DSN '04)*. 125--134.

[43] Yingxun Fu and Jiwu Shu. 2015. D-Code: An efficient RAID-6 code to optimize I/O loads and read performance. In *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS '15)*. 603--612.

[44] Atul Goel and Peter Corbett. 2012. RAID triple parity. *ACM SIGOPS Operating Systems Review* 46, 3 (2012), 41--49.

[45] Guowen Gong, Zhirong Shen, Suzhen Wu, Xiaolu Li, and Patrick P. C. Lee. 2021. Optimal rack-coordinated updates in erasure-coded data centers. In *Proceeding of the 2021 IEEE Conference on Computer Communications (INFOCOM '21)*. 1--10.

[46] Garth R. Goodson, Jay J. Wylie, Gregory R. Ganger, and Michael K. Reiter. 2004. Efficient Byzantine-tolerant erasure-coded storage. In *Proceeding of the 2004 International Conference on Dependable Systems and Networks (DSN '04)*. 135--144.

[47] Kevin M Greenan. 2009. *Reliability and power-efficiency in erasure-coded storage systems.* Technical Report. University of California, Santa Cruz.

[48] Kevin M. Greenan, Ethan L. Miller, and Thomas J. E. Schwarz. 2008. Optimizing Galois Field Arithmetic for Diverse Processor Architectures and Applications. In *Proceedings of the 16th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCTOS '08)*. 257--266.

[49] Kevin M. Greenan, Ethan L. Miller, and Jay J. Wylie. 2008. Reliability of flat XOR-based erasure codes on heterogeneous devices. In *Proceedings of 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN '08)*. 147--156.

[50] Kevin M. Greenan, James S. Plank, and Jay J. Wylie. 2010. Mean time to meaningless: MTTDL, Markov models, and storage system reliability. In *Proceedings of the 2nd Workshop on Hot Topics in Storage and File Systems (HotStorage '10)*.

[51] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2017. Efficient memory disaggregation with infiniswap. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI '17)*. 649--667.

[52] Andreas Haeberlen and Alan Mislove. 2005. Glacier: Highly Durable, Decentralized Storage Despite Massive Correlated Failures. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*. 143--158.

[53] James Lee Hafner. 2005. WEAVER Codes: Highly fault tolerant erasure codes for storage systems. In *Proceendings of the 4th USENIX Conference on File and Storage Technologies (FAST '05)*.

[54] James Lee Hafner. 2006. HoVer Erasure Codes For Disk Arrays. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*. 217--226.

[55] James Lee Hafner, Veera Deenadhayalan, KK Rao, and John A Tomlin. 2005. Matrix Methods for Lost Data Reconstruction in Erasure Codes. In *Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies (FAST'05)*. 15--30.

[56] Raluca Halalai, Pascal Felber, Anne-Marie Kermarrec, and François Taïani. 2017. Agar: A Caching System for Erasure-Coded Data. In *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS '17)*. 23--33.

[57] Robert J Hall. 2016. Tools for predicting the reliability of large-scale storage systems. *ACM Transactions on Storage* 12, 4 (2016), 1--30.

[58] Shujie Han, Patrick PC Lee, Fan Xu, Yi Liu, Cheng He, and Jiongzhou Liu. 2021. An In-Depth Study of Correlated Failures in Production SSD-Based Data Centers. In *Proceendings of the 19th USENIX Conference on File and Storage Technologies (FAST '21)*. 417--429.

[59] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A. Chien, and Haryadi S. Gunawi. 2016. The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*. 263--276.

[60] Mark Holland, Garth A. Gibson, and Daniel P. Siewiorek. 1994. Architectures and Algorithms for On-line Failure Recovery in Redundant Disk Arrays. *Distributed Parallel Databases* 2, 3 (1994), 295--335.

[61] Yuchong Hu, Liangfeng Cheng, Qiaori Yao, Patrick P. C. Lee, Weichun Wang, and Wei Chen. 2021. Exploiting combined locality for wide-stripe erasure coding in distributed storage. In *Proceeding of the 19th USENIX Conference on File and Storage Technologies (FAST '21)*. 233--248.

[62] Yuchong Hu, Xiaolu Li, Mi Zhang, Patrick P. C. Lee, Xiaoyang Zhang, Pan Zhou, and Dan Feng. 2017. Optimal Repair Layering for Erasure-Coded Data Centers: From Theory to Practice. *ACM Transactions on Storage* 13, 4 (2017), 1--24.

[63] Yaochen Hu, Yushi Wang, Bang Liu, Di Niu, and Cheng Huang. 2017. Latency reduction and load balancing in coded storage systems. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC '17)*. 365--377.

[64] Yuchong Hu, Xiaoyang Zhang, Patrick P. C. Lee, and Pan Zhou. 2022. NCScale: Toward optimal storage scaling via network coding. *IEEE/ACM Transactions on Networking* 30, 1 (2022), 271--284.

[65] Cheng Huang, Minghua Chen, and Jin Li. 2013. Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems. *ACM Transactions on Storage* 9, 1 (2013), 1--28.
[66] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brads Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. 2012. Erasure coding in Windows Azure Storage. In *Proceedings of 2012 USENIX Annual Technical Conference (USENIX ATC '12)*. 15--26.
[67] Cheng Huang and Lihao Xu. 2005. STAR: An efficient coding scheme for correcting triple storage node failures. In *Proceedings of 4th USENIX Conference on File and Storage Technologies (FAST '05)*.
[68] Jianzhong Huang, Xianhai Liang, Xiao Qin, Qiang Cao, and Changsheng Xie. 2015. PUSH: A pipelined reconstruction I/O for erasure-coded storage clusters. *IEEE Transactions on Parallel and Distributed Systems* 26, 2 (2015), 516--526.
[69] Jianzhong Huang, Xianhai Liang, Xiao Qin, Ping Xie, and Changsheng Xie. 2015. Scale-RS: An Efficient Scaling Scheme for RS-Coded Storage Clusters. *IEEE Transactions on Parallel and Distributed Systems* 26, 6 (2015), 1704--1717.
[70] Lingchen Huang, Huazi Zhang, Rong Li, Yiqun Ge, and Jun Wang. 2019. AI coding: Learning to construct error correction codes. *IEEE Transactions on Communications* 68, 1 (2019), 26--39.
[71] Ilias Iliadis. 2023. Reliability Evaluation of Erasure-coded Storage Systems with Latent Errors. *ACM Transactions on Storage* 19, 1 (2023), 1--47.
[72] Ilias Iliadis, Robert Haas, Xiao-Yu Hu, and Evangelos Eleftheriou. 2011. Disk scrubbing versus intradisk redundancy for RAID storage systems. *ACM Transactions on Storage* 7, 2 (2011), 1--42.
[73] Ilias Iliadis and Vinodh Venkatesan. 2015. Rebuttal to Beyond MTTDL: A closed-form RAID-6 reliability equation. *ACM Transactions on Storage* 11, 2 (2015), 1--10.
[74] Shehbaz Jaffer, Kaveh Mahdaviani, and Bianca Schroeder. 2020. Rethinking WOM codes to enhance the lifetime in new SSD generations. In *Proceedings of the 12th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '20)*. 1--8.
[75] Shehbaz Jaffer, Kaveh Mahdaviani, and Bianca Schroeder. 2022. Improving the reliability of next generation SSDs using WOM-v codes. In *Proceedings of the 20th USENIX Conference on File and Storage Technologies (FAST '22)*. 117--132.
[76] Jerasure. Accessed in March 2024. Jerasure: Erasure Coding Library. https://jerasure.org/.
[77] Yulei Jia, Guangping Xu, Chi Wan Sung, Salwa Mostafa, and Yulei Wu. 2022. HRaft: Adaptive erasure coded data maintenance for consensus in distributed networks. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS '22)*. 1316--1326.
[78] Wei Jiang, Hao Jiang, Jing Wu, and Qimei Chen. 2023. Accelerating distributed cloud storage systems with in-network computing. *IEEE Network* 37, 4 (2023), 64--70.
[79] Wei Jiang, Hao Jiang, Jing Wu, and Pengcheng Zhou. 2023. Accelerating network coding with programmable switch ASICs. In *Proceedings of the 2023 IEEE International Conference on Communications (ICC '23)*. 1093--1099.
[80] Chao Jin, Dan Feng, Hong Jiang, and Lei Tian. 2011. RAID6L: A log-assisted RAID6 storage architecture with improved write performance. In *Proceedings of the 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST '11)*. 1--6.
[81] Hai Jin, Ruikun Luo, Qiang He, Song Wu, Zilai Zeng, and Xiaoyu Xia. 2023. Cost-effective data placement in edge storage systems with erasure code. *IEEE Transactions on Services Computing* 16, 2 (2023), 1039--1050.
[82] Saurabh Kadekodi, Francisco Maturana, Sanjith Athlur, Arif Merchant, K. V. Rashmi, and Gregory R. Ganger. 2022. Tiger: Disk-adaptive redundancy without placement restrictions. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*. 413--429.
[83] Saurabh Kadekodi, Francisco Maturana, Suhas Jayaram Subramanya, Juncheng Yang, K. V. Rashmi, and Gregory R. Ganger. 2020. PACEMAKER: Avoiding HeART attacks in storage clusters with disk-adaptive redundancy. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*. 369--385.
[84] Saurabh Kadekodi, K. V. Rashmi, and Gregory R. Ganger. 2019. Cluster storage systems gotta have HeART: Improving storage efficiency by exploiting disk-reliability heterogeneity. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST '19)*. 345--358.
[85] Saurabh Kadekodi, Shashwat Silas, David Clausen, and Arif Merchant. 2023. Practical design considerations for wide locally recoverable codes (LRCs). In *Proceeding of the 21st USENIX Conference on File and Storage Technologies (FAST '23)*. 1--16.
[86] Osama Khan, Randal Burns, James Plank, William Pierce, and Cheng Huang. 2012. Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST '12)*. 251--264.
[87] Oleg Kolosov, Gala Yadgar, Matan Liram, Itzhak Tamo, and Alexander Barg. 2018. On fault tolerance, locality, and optimality in locally repairable codes. In *Proceeding of the 2018 USENIX Annual Technical Conference (ATC'18)*. 865--877.

[88] Kishori M Konwar, N Prakash, Nancy Lynch, and Muriel Médard. 2017. A layered architecture for erasure-coded consistent distributed storage. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'17)*. 63--72.
[89] Jack Kosaian, KV Rashmi, and Shivaram Venkataraman. 2019. Parity models: erasure-coded resilience for prediction serving systems. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP '19)*. 30--46.
[90] Ramakrishna Kotla, Lorenzo Alvisi, and Mike Dahlin. 2007. SafeStore: A Durable and Practical Storage System. In *Proceedings of the 2007 USENIX Annual Technical Conference (ATC'07)*. 129--142.
[91] Katina Kralevska, Danilo Gligoroski, Rune E Jensen, and Harald Øverby. 2018. Hashtag erasure codes: From theory to practice. *IEEE Transactions on Big Data* 4, 4 (2018), 516--529.
[92] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. 2000. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS IX)*. 190--201.
[93] Chunbo Lai, Song Jiang, Liqiong Yang, Shiding Lin, Guangyu Sun, Zhenyu Hou, Can Cui, and Jason Cong. 2015. Atlas: Baidu's key-value storage system for cloud data. In *Proceedings of the 31st Symposium on Mass Storage Systems and Technologies (MSST '15)*. 1--14.
[94] Leslie Lamport. 1998. The part-time parliament. *ACM Transactions on Computer Systems* 16, 2 (1998), 133–--169.
[95] Leslie Lamport. 2001. Paxos made simple. *ACM Transactions on SIGACT News* 32, 4 (2001), 51--58.
[96] Youngmoon Lee, Hasan Al Maruf, Mosharaf Chowdhury, Asaf Cidon, and Kang G. Shin. 2022. Hydra: Resilient and highly available remote memory. In *Proceedings of 20th USENIX Conference on File and Storage Technologies (FAST '22)*. 181--198.
[97] Michael Ley. [n. d.]. The DBLP Computer Science Bibliography. https://dblp.org. Retrieved in Aug 2024.
[98] Huiba Li, Yiming Zhang, Zhiming Zhang, Shengyun Liu, Dongsheng Li, Xiaohui Liu, and Yuxing Peng. 2017. PARIX: Speculative partial writes in erasure-coded systems. In *Proceedings of the 15th USENIX Annual Technical Conference (ATC '17)*. 581--587.
[99] Jun Li and Baochun Li. 2013. Erasure Coding for Cloud Storage Systems: A Survey. *Tsinghua Science and Technology* 18, 3 (2013), 259--272.
[100] Mingqiang Li and Patrick P. C. Lee. 2014. STAIR Codes: A general family of erasure codes for tolerating device and sector failures. *ACM Transactions on Storage* 10, 4 (2014), 1553--3077.
[101] Mingqiang Li and Patrick P. C. Lee. 2018. Relieving Both Storage and Recovery Burdens in Big Data Clusters with R-STAIR Codes. *IEEE Internet Computing* 22, 4 (2018), 15--26.
[102] Mingqiang Li, Jiwu Shu, and Weimin Zheng. 2009. GRID codes: Strip-based erasure codes with high fault tolerance for storage systems. *ACM Transactions on Storage* 4, 4 (2009), 1--22.
[103] Qiliang Li, Liangliang Xu, Yongkun Li, Min Lyu, Wei Wang, Pengfei Zuo, and Yinlong Xu. 2024. Enabling efficient erasure coding in disaggregated memory systems. *IEEE Transactions on Parallel and Distributed Systems* 35, 01 (2024), 154--168.
[104] Runhui Li, Yuchong Hu, and Patrick P. C. Lee. 2017. Enabling efficient and reliable transition from replication to erasure coding for clustered file systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 9 (2017), 2500--2513.
[105] Runhui Li, Patrick P. C. Lee, and Yuchong Hu. 2014. Degraded-first scheduling for MapReduce in erasure-coded storage clusters. In *Proceedings of 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '14)*. 419--430.
[106] Runhui Li, Xiaolu Li, Patrick P. C. Lee, and Qun Huang. 2017. Repair pipelining for erasure-coded storage. In *Proceedings of 2017 USENIX Annual Technical Conference (USENIX ATC '17)*. 567--579.
[107] Runhui Li, Jian Lin, and Patrick P. C. Lee. 2015. Enabling Concurrent Failure Recovery for Regenerating-Coding-Based Storage Systems: From Theory to Practice. *IEEE Trans. Comput.* 64, 7 (2015), 1898--1911.
[108] Shiyi Li, Qiang Cao, Lei Tian, Shenggang Wan, Lu Qian, and Changsheng Xie. 2015. PSG-Codes: An erasure codes family with high fault tolerance and fast recovery. In *Proceedings of the 34th IEEE Symposium on Reliable Distributed Systems (SRDS'15)*. 47--57.
[109] Shiyi Li, Qiang Cao, Shenggang Wan, Wen Xia, and Changsheng Xie. 2023. gPPM: A generalized matrix operation and parallel algorithm to accelerate the encoding/decoding process of erasure codes. *ACM Transactions On Architecture And Code Optimization* 20, 4 (2023), 1--25.
[110] Shenglong Li, Quanlu Zhang, Zhi Yang, and Yafei Dai. 2017. BCStore: Bandwidth-efficient in-memory KV-store with batch coding. In *Proceedings of the 31th International Conference on Massive Storage Systems and Technology (MSST '17)*. 1--13.
[111] Xiaolu Li, Keyun Cheng, Zhirong Shen, and Patrick P. C. Lee. 2022. Fast proactive repair in erasure-coded storage: Analysis, design, and implementation. *IEEE Transactions on Parallel and Distributed Systems* 33, 12 (2022), 3400--3414.

[112] Xiaolu Li, Keyun Cheng, Kaicheng Tang, Patrick P. C. Lee, Yuchong Hu, Dan Feng, Jie Li, and Ting-Yi Wu. 2023. ParaRC: Embracing sub-packetization for repair parallelization in MSR-coded storage. In *Proceedings of 21st USENIX Conference on File and Storage Technologies (FAST '23)*. 17--32.

[113] Xiaolu Li, Runhui Li, Patrick P. C. Lee, and Yuchong Hu. 2019. OpenEC: Toward Unified and Configurable Erasure Coding Management in Distributed Storage Systems. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST '19)*. 331--344.

[114] Xiaolu Li, Zuoru Yang, Jinhong Li, Runhui Li, Patrick P. C. Lee, Qun Huang, and Yuchong Hu. 2021. Repair Pipelining for Erasure-Coded Storage: Algorithms and Evaluation. *ACM Transactions on Storage* 17, 2 (2021), 13:1--13:29.

[115] Yin Li, Hao Wang, Xuebin Zhang, Ning Zheng, Shafa Dahandeh, and Tong Zhang. 2017. Facilitating magnetic recording technology scaling for data center hard disk drives through filesystem-level transparent local erasure coding. In *Proceeding of the 15th USENIX Conference on File and Storage Technologies (FAST '17)*. 135--148.

[116] Shiyao Lin, Guowen Gong, Zhirong Shen, Patrick P. C. Lee, and Jiwu Shu. 2021. Boosting full-node repair in erasure-coded storage. In *Proceedings of 2021 USENIX Annual Technical Conference (USENIX ATC '21)*. 641--655.

[117] Witold Litwin, Rim Moussa, and Thomas Schwarz. 2005. LH^*_{RS} - A Highly-Available Scalable Distributed Data Structure. *ACM Trans. Database Syst.* 30, 3 (2005), 769--811.

[118] Kaiyang Liu, Jun Peng, Jingrong Wang, Zhiwu Huang, and Jianping Pan. 2023. Adaptive and scalable caching with erasure codes in distributed cloud-edge storage systems. *IEEE Transactions on Cloud Computing* 11, 2 (2023), 1840--1853.

[119] Shiqiu Liu and Frédérique Oggier. 2018. An Overview of Coding for Distributed Storage Systems. *Network Coding and Subspace Designs* (2018), 363--383.

[120] Sidi Lu, Bing Luo, Tirthak Patel, Yongtao Yao, Devesh Tiwari, and Weisong Shi. 2020. Making disk failure predictions SMARTer. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20)*. 151--167.

[121] Michael Luby, Roberto Padovani, Thomas J. Richardson, Lorenz Minder, and Pooja Aggarwal. 2019. Liquid cloud storage. *ACM Transactions on Storage* 15, 1 (2019), 1--49.

[122] Jianqiang Luo, Mochan Shrestha, Lihao Xu, and James S. Plank. 2014. Efficient encoding schedules for XOR-based erasure codes. *IEEE Trans. Comput.* 63, 9 (2014), 2259--2272.

[123] Fabio Margaglia, Gala Yadgar, Eitan Yaakobi, Yue Li, Assaf Schuster, and Andr Brinkmann. 2016. The devil is in the details: Implementing flash page reuse with WOM codes. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16)*. 95--109.

[124] Francisco Maturana and K. V. Rashmi. 2022. Convertible codes: Enabling efficient conversion of coded data in distributed storage. *IEEE Transactions on Information Theory* 68, 7 (2022), 4392--4407.

[125] Francisco Maturana and K. V. Rashmi. 2023. Bandwidth cost of code conversions in distributed storage: Fundamental limits and optimal constructions. *IEEE Transactions on Information Theory* 69, 8 (2023), 4993--5008.

[126] Subrata Mitra, Rajesh Panta, Moo-Ryong Ra, and Saurabh Bagchi. 2016. Partial-Parallel-Repair (PPR): A distributed technique for repairing erasure coded storage. In *Proceedings of the 11th European Conference on Computer Systems (EuroSys '16)*. 1--16.

[127] Shuai Mu, Kang Chen, Yongwei Wu, and Weimin Zheng. 2014. When PAXOS meets erasure code: Reduce network and storage cost in state machine replication. In *Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '14)*. 61–72.

[128] Lev Mukhanov, Konstantinos Tovletoglou, Hans Vandierendonck, Dimitrios S Nikolopoulos, and Georgios Karakonstantis. 2019. Workload-aware dram error prediction using machine learning. In *Proceedings of IEEE International Symposium on Workload Characterization (IISWC) (IISWC'19)*. 106--118.

[129] Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang, and Sanjeev Kumar. 2014. f4: Facebook's Warm BLOB Storage System. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*. 383--398.

[130] Andrea C. Arpaci-Dusseau Muthian Sivathanu, Vijayan Prabhakaran and Remzi H. Arpaci-Dusseau. 2004. Improving Storage System Availability with D-GRAID. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST '04)*. 15--30.

[131] Suman Nath, Haifeng Yu, Phillip B. Gibbons, and Srinivasan Seshan. 2006. Subtleties in Tolerating Correlated Failures in Wide-area Storage Systems. In *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI'06)*. 225--238.

[132] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceeding of the 2014 USENIX Annual Technical Conference (ATC'14)*. 305--319.

[133] Michael Ovsiannikov, Silvius Rus, Damian Reeves, Paul Sutter, Sriram Rao, and Jim Kelly. 2013. The Quantcast file system. *Proceedings of the VLDB Endowment* 6, 11 (2013), 1092--1101.

[134] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *Journal of Algorithms* 51, 2 (2004), 122--144.

[135] Lluis Pamies-Juarez, Filip Blagojevi, Robert Mateescu, Cyril Gyuot, Eyal En Gad, and Zvonimir Bandi. 2016. Opening the Chrysalis: On the real repair performance of MSR codes. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16)*. 81--94.

[136] Lluis Pamies-Juarez, Anwitaman Datta, and Frederique Oggier. 2013. RapidRAID: Pipelined erasure codes for fast data archival in distributed storage systems. In *Proceedings of the IEEE INFOCOM 2013 - IEEE Conference on Computer Communications (INFOCOM '13)*. 1294--1302.

[137] Dimitris S. Papailiopoulos, Jianqiang Luo, Alexandros G. Dimakis, Cheng Huang, and Jin Li. 2012. Simple regenerating codes: Network coding for cloud storage. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'12)*. 2801--2805. https://doi.org/10.1109/INFCOM.2012.6195703

[138] David A. Patterson, Garth Gibson, and Randy H. Katz. 1988. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data (SIGMOD '88)*. 109–--116.

[139] Xiaoqiang Pei, Yijie Wang, Xingkong Ma, and Fangliang Xu. 2016. T-Update: A tree-structured update scheme with top-down transmission in erasure-coded systems. In *Proceeding of the 2016 IEEE International Conference on Computer Communications (INFOCOM '16)*. 1--9.

[140] Xiaoqiang Pei, Yijie Wang, Xingkong Ma, and Fangliang Xu. 2017. Efficient in-place update with grouped and pipelined data transmission in erasure-coded storage systems. *Future Generation Computer Systems* 69 (2017), 24--40.

[141] James S. Plank. 2008. The RAID-6 Liberation Codes. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08)*. 97--110.

[142] James S. Plank and Mario Blaum. 2014. Sector-Disk (SD) Erasure Codes for Mixed Failure Modes in RAID Systems. *ACM Transactions on Storage* 10, 1 (2014), 1--17.

[143] James S. Plank, Kevin M. Greenan, and Ethan L. Miller. 2013. Screaming fast Galois field arithmetic using Intel SIMD instructions. In *Proceedings of 12th USENIX Conference on File and Storage Technologies (FAST '13)*. 299--306.

[144] James S. Plank and Cheng Huang. 2013. Tutorial: Erasure Coding for Storage Applications. Slides presented at USENIX FAST 2013.

[145] James S Plank, Jianqiang Luo, Catherine D Schuman, Lihao Xu, Zooko Wilcox-O'Hearn, et al. 2009. A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries for Storage.. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST '09)*. 253--265.

[146] James S. Plank, Catherine D. Schuman, and B. Devin Robison. 2012. Heuristics for optimizing matrix-based erasure codes for fault-tolerant storage systems. In *Proceedings of IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '12)*. 1--12.

[147] Pure Storage. 2023. The Data Center of the Future Is All-flash. https://blog.purestorage.com/purely-educational/the-data-center-of-the-future-is-all-flash/.

[148] Yi Qiao, Xiao Kong, Menghao Zhang, Yu Zhou, Mingwei Xu, and Jun Bi. 2020. Towards in-network acceleration of erasure coding. In *Proceedings of the Symposium on SDN Research (SOSR '20)*. 41–--47.

[149] Vinayak Ramkumar, S. B. Balaji, Birenjith Sasidharan, Myna Vajha, M. Nikhil Krishnan, and P. Vijay Kumar. 2022. Codes for Distributed Storage. *Foundations and Trends® in Communications and Information Theory* 19, 4 (2022), 547--813.

[150] KK Rao, James Lee Hafner, and Richard A Golding. 2010. Reliability for networked storage nodes. *IEEE Transactions on Dependable and Secure Computing* 8, 3 (2010), 404--418.

[151] K. V. Rashmi, Mosharaf Chowdhury, Jack Kosaian, Ion Stoica, and Kannan Ramchandran. 2016. EC-Cache: Load-balanced, low-latency cluster caching with online erasure coding. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*. 401--417.

[152] K. V. Rashmi, Preetum Nakkiran, Jingyan Wang, Nihar B. Shah, and Kannan Ramchandran. 2015. Having your cake and eating it too: Jointly optimal erasure codes for I/O, storage, and network-bandwidth. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (DSN '10)*. 81--94.

[153] K. V. Rashmi, Nihar B. Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. 2013. A Solution to the Network Challenges of Data Recovery in Erasure-coded Distributed Storage Systems: A Study on the Facebook Warehouse Cluster. In *Proceedings of the 5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'13)*. 8:1--8:5.

[154] K. V. Rashmi, Nihar B. Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. 2014. A "hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14)*. 331–--342.

[155] K. V. Rashmi, Nihar B. Shah, and P. Vijay Kumar. 2011. Optimal Exact-regenerating Codes for Distributed Storage at the MSR and MBR Points via a Product-matrix Construction. *IEEE Transactions on Information Theory* 57, 8 (Aug 2011), 5227--5239.

[156] K. V. Rashmi, Nihar B Shah, and Kannan Ramchandran. 2017. A piggybacking design framework for read-and download-efficient distributed storage codes. *IEEE Transactions on Information Theory* 63, 9 (2017), 5802--5820.

[157] I.S. Reed and G. Solomon. 1960. Polynomial Codes over Certain Finite Fields. *J. Soc. Indust. Appl. Math.* 8, 2 (1960), 300--304.

[158] David Reinsel, John Gantz, and John Rydnin. 2018. The Digitization of the World From Edge to Core. https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf. International Data Corporation Doc. #US44413318.

[159] Yanjing Ren, Yuanming Ren, Xiaolu Li, Yuchong Hu, Jingwei Li, and Patrick P. C. Lee. 2024. ELECT: Enabling erasure coding tiering for LSM-tree-based storage. In *Proceedings of the 22nd USENIX Conference on File and Storage Technologies (FATS'24)*. 293--310.

[160] Rodrigo Rodrigues and Barbara Liskov. 2005. High Availability in DHTs: Erasure Coding vs. Replication. In *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS'05)*. 226--239.

[161] Eitan Rosenfeld, Aviad Zuck, Nadav Amit, Michael Factor, and Dan Tsafrir. 2020. RAIDP: replication with intra-disk parity. In *Proceedings of the 15th European Conference on Computer Systems (EuroSys'20)*. 26:1--26:17.

[162] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. 2013. XORing Elephants: Novel erasure codes for big data. *Proceedings of the VLDB Endowment* 6, 5 (2013), 325--336.

[163] Nihar B. Shah, K. V. Rashmi, P. Vijay Kumar, and Kannan Ramchandran. 2012. Interference Alignment in Regenerating Codes for Distributed Storage: Necessity and Code Constructions. *IEEE Transactions on Information Theory* 58, 4 (Sep 2012), 2134--2158.

[164] Yingdi Shan, Kang Chen, Tuoyu Gong, Lidong Zhou, Tai Zhou, and Yongwei Wu. 2021. Geometric Partitioning: Explore the boundary of optimal erasure code repair. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21)*. 457---471.

[165] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. LegoOS: A disseminated, distributed OS for hardware resource disaggregation. In *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI '18)*. 69--87.

[166] Dipti Shankar, Xiaoyi Lu, and Dhabaleswar K. Panda. 2017. High-performance and resilient key-value store with online erasure coding for big data workloads. In *Proceedings of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS '17)*. 527--537.

[167] Jiajie Shen, Kai Zhang, Jiazhen Gu, Yangfan Zhou, and Xin Wang. 2018. Efficient scheduling for multi-block updates in erasure coding based storage systems. *IEEE Transactions on Computers* 67, 4 (2018), 573--581.

[168] Zhirong Shen, Patrick P.C. Lee, Jiwu Shu, and Wenzhong Guo. 2017. Correlation-aware stripe organization for efficient writes in erasure-coded storage systems. In *Proceeding of the 2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS '17)*. 134--143.

[169] Zhirong Shen and Patrick P. C. Lee. 2018. Cross-rack-aware updates in erasure-coded data centers. In *Proc. of the 47th International Conference on Parallel Processing (ICPP 18)*. 1--10.

[170] Zhirong Shen, Jiwu Shu, and Yingxun Fu. 2016. HV Code: An all-around MDS code for RAID-6 storage systems. *IEEE Transactions on Parallel and Distributed Systems* 27, 6 (2016), 1674--1686.

[171] Zhirong Shen, Jiwu Shu, and Yingxun Fu. 2016. Parity-switched data placement: Optimizing partial stripe writes in XOR-coded storage systems. *IEEE Transactions on Parallel and Distributed Systems* 27, 11 (2016), 3311--3322.

[172] Zhirong Shen, Jiwu Shu, Zhijie Huang, and Yingxun Fu. 2020. ClusterSR: Cluster-aware scattered repair in erasure-coded storage. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS '20)*. 42--51.

[173] Zhirong Shen, Jiwu Shu, and Patrick P. C. Lee. 2016. Reconsidering single failure recovery in clustered file systems. In *Proceedings of 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '16)*. 323--334.

[174] Zhirong Shen, Jiwu Shu, Patrick P. C. Lee, and Yingxun Fu. 2017. Seek-Efficient I/O Optimization in Single Failure Recovery for XOR-Coded Storage Systems. *IEEE Transactions on Parallel and Distributed Systems* 28, 3 (2017), 877--890.

[175] Mark Silberstein, Lakshmi Ganesh, Yang Wang, Lorenzo Alvisi, and Mike Dahlin. 2014. Lazy means smart: Reducing repair bandwidth costs in erasure-coded distributed storage. In *Proceedings of the 7th ACM International Systems and Storage Conference (SYSTOR '14)*. 1--7.

[176] Daniel Stodolsky, Garth Gibson, and Mark Holland. 1993. Parity logging overcoming the small write problem in redundant disk arrays. In *Proceedings of the 20th Annual International Symposium on Computer Architecture (ISCA '93)*. 64--75.

[177] Changho Suh and Kannan Ramchandran. 2011. Exact-Repair MDS Code Construction Using Interference Alignment. *IEEE Transactions on Information Theory* 57, 3 (Mar 2011), 1425--1442.

[178] Itzhak Tamo and Alexander Barg. 2014. A family of optimal Locally Recoverable Codes. *IEEE Transactions on Information Theory* 60, 8 (2014), 4661--4676.

[179] Kaicheng Tang, Keyun Cheng, Helen H. W. Chan, Xiaolu Li, Patrick P. C. Lee, Yuchong Hu, Jie Li, and Ting-Yi Wu. 2023. Balancing Repair Bandwidth and Sub-Packetization in Erasure-Coded Storage via Elastic Transformation.

In *Proceedings of IEEE Conference on Computer Communications (INFOCOM '23)*. 1--10. https://doi.org/10.1109/INFOCOM53939.2023.10228984

[180] Konstantin Taranov, Gustavo Alonso, and Torsten Hoefler. 2018. Fast and strongly-consistent per-item resilience in key-value stores. In *Proceedings of the 13th European Conference on Computer Systems (EuroSys '18)*. 1--14.

[181] Petroc Taylor. 2023. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025. https://www.statista.com/statistics/871513/worldwide-data-created/. Statista.

[182] Alexander Thomasian. 2005. Reconstruct Versus Read-modify Writes in RAID. *Inform. Process. Lett.* 93, 4 (2005), 163--168.

[183] Alexander Thomasian and Mario Blaum. 2009. Higher reliability redundant disk arrays: Organization, operation, and coding. *ACM Transactions on Storage* 5, 3 (2009), 1--59.

[184] Yuya Uezato. 2021. Accelerating XOR-based erasure coding using program optimization techniques. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '21)*. 1--14.

[185] Muhammed Uluyol, Anthony Huang, Ayush Goel, Mosharaf Chowdhury, and Harsha V. Madhyastha. 2020. Near-Optimal Latency Versus Cost Tradeoffs in Geo-Distributed Storage. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20)*. 157--180.

[186] Myna Vajha, Vinayak Ramkumar, Bhagyashree Puranik, Ganesh Kini, Elita Lobo, Birenjith Sasidharan, P. Vijay Kumar, Alexandar Barg, Min Ye, Srinivasan Narayanamurthy, Syed Hussain, and Siddhartha Nandi. 2018. Clay Codes: Moulding MDS codes to yield an MSR code. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST '18)*. 139--154.

[187] Ao Wang, Jingyuan Zhang, Xiaolong Ma, Ali Anwar, Lukas Rupprecht, Dimitrios Skourtis, Vasily Tarasov, Feng Yan, and Yue Cheng. 2020. InfiniCache: Exploiting ephemeral serverless functions to build a cost-effective memory cache. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST '20)*. 267--281.

[188] Chenxi Wang, Haoran Ma, Shi Liu, Yuanqi Li, Zhenyuan Ruan, Khanh Nguyen, Michael D. Bond, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. 2020. Semeru: A memory-disaggregated managed runtime. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI '20)*. 261--280.

[189] Fang Wang, Yingjie Tang, Yanwen Xie, and Xuehai Tang. 2019. XORInc: Optimizing data repair and update for erasure-coded systems with XOR-based in-network computation. In *Proceedings of the 35th Symposium on Mass Storage Systems and Technologies (MSST '19)*. 244--256.

[190] Meng Wang, Jiajun Mao, Rajdeep Rana, John Bent, Serkay Olmez, Anjus George, Garrett Wilson Ransom, Jun Li, and Haryadi S. Gunawi. 2023. Design considerations and analysis of multi-level erasure coding in large-scale data centers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23)*. 1--13.

[191] Neng Wang, Yinlong Xu, Yongkun Li, and Si Wu. 2016. OI-RAID: A two-layer RAID architecture towards fast recovery and high reliability. In *Proceedings of 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '16)*. 61--72.

[192] Qing Wang, Youyou Lu, and Jiwu Shu. 2022. Sherman: A write-optimized distributed b+Tree index on disaggregated Memory. In *Proceedings of SIGMOD '22: International Conference on Management of Data (SIGMOD '22)*. 1033--1048.

[193] Zizhong Wang, Tongliang Li, Haixia Wang, Airan Shao, Yunren Bai, Shangming Cai, Zihan Xu, and Dongsheng Wang. 2020. CRaft: An erasure-coding-supported version of Raft for reducing storage cost and network cost. In *18th USENIX Conference on File and Storage Technologies (FAST '20)*. 297--308.

[194] Hakim Weatherspoon and John D Kubiatowicz. 2002. Erasure coding vs. replication: A quantitative comparison. In *Proceedings of International Workshop on Peer-to-Peer Systems (IPTPS'02)*. 328--337.

[195] James Westall and James Martin. 2010. *An Introduction to Galois Fields and Reed-Solomon Coding*. Technical Report. School of Computing, Clemson University. https://people.computing.clemson.edu/~jmarty/papers/IntroToGaloisFieldsAndRSCoding.pdf, Retrieved in Aug 2024..

[196] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. 1996. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems* 14, 1 (1996), 108--136.

[197] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. 2021. Demand-Aware Erasure Coding for Distributed Storage Systems. *IEEE Transactions on Cloud Computing* 9, 2 (2021), 532--545.

[198] Chentao Wu and Xubin He. 2012. GSR: A global stripe-based redistribution approach to accelerate RAID-5 scaling. In *Proceedings of the 41st International Conference on Parallel Processing (ICPP '12)*. 460--469.

[199] Chentao Wu and Xubin He. 2013. A Flexible Framework to Enhance RAID-6 Scalability via Exploiting the Similarities among MDS Codes. In *Proceedings of the 42nd International Conference on Parallel Processing (ICPP '13)*. 542--551.

[200] Chentao Wu, Xubin He, Jizhong Han, Huailiang Tan, and Changsheng Xie. 2012. SDM: A Stripe-Based Data Migration Scheme to Improve the Scalability of RAID-6. In *Proceedings of the 2012 IEEE International Conference on Cluster Computing (CLUSTER '12)*. 542--551.

[201] Chentao Wu, Xubin He, Guanying Wu, Shenggang Wan, Xiaohua Liu, Qiang Cao, and Changsheng Xie. 2011. HDP code: A Horizontal-Diagonal Parity Code to Optimize I/O load balancing in RAID-6. In *Proceedings of the 41st IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'11)*. 209--220.

[202] Chentao Wu, Shenggang Wan, Xubin He, Qiang Cao, and Changsheng Xie. 2011. H-Code: A Hybrid MDS Array Code to Optimize Partial Stripe Writes in RAID-6. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'11)*. 782--793.

[203] Si Wu, Qingpeng Du, Patrick P. C. Lee, Yongkun Li, and Yinlong Xu. 2022. Optimal data placement for stripe merging in locally repairable codes. In *Proceedings of the 2022 IEEE Conference on Computer Communications (INFOCOM '22)*. 1669--1678.

[204] Si Wu, Zhirong Shen, Patrick PC Lee, Zhiwei Bai, and Yinlong Xu. 2024. Elastic Reed-Solomon codes for efficient redundancy transitioning in distributed key-value stores. *IEEE/ACM Transactions on Networking* 32, 1 (2024), 670--685.

[205] Si Wu, Zhirong Shen, and Patrick P. C. Lee. 2020. On the optimal repair-scaling trade-off in Locally Repairable Codes. In *Proceedings of the 2020 IEEE Conference on Computer Communications (INFOCOM '20)*. 2155--2164.

[206] Si Wu, Yinlong Xu, Yongkun Li, and Zhijia Yang. 2016. I/O-efficient scaling schemes for distributed storage systems with CRS codes. *IEEE Transactions on Parallel and Distributed Systems* 27, 9 (2016), 2639--2652.

[207] Junxu Xia, Deke Guo, and Geyao Cheng. 2019. In-network block repairing for erasure coding storage systems. *Concurrency and Computation: Practice and Experience* 31, 24 (2019), e5432.

[208] Jie Xia, Jianzhong Huang, Xiao Qin, Qiang Cao, and Changsheng Xie. 2017. Revisiting updating schemes for erasure-coded in-memory stores. In *Proceedings of the 2017 International Conference on Networking, Architecture, and Storage (NAS '17)*. 1--6.

[209] Junxu Xia, Lailong Luo, Bowen Sun, Geyao Cheng, and Deke Guo. 2024. Parallelized In-Network Aggregation for Failure Repair in Erasure-Coded Storage Systems. *IEEE/ACM Transactions on Networking* 01 (2024), 1--16.

[210] Mingyuan Xia, Mohit Saxena, Mario Blaum, and David A. Pease. 2015. A tale of two erasure codes in HDFS. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST '15)*. 213--226.

[211] Liping Xiang, Yinlong Xu, John C.S. Lui, Qian Chang, Yubiao Pan, and Runhui Li. 2011. A Hybrid Approach of Failed Disk Recovery Using RAID-6 Codes: Algorithms and Performance Evaluation. *ACM Transactions on Storage* 7, 3 (2011).

[212] Qin Xin, Ethan L Miller, and SJ Thomas JE Schwarz. 2004. Evaluation of distributed recovery in large-scale storage systems. In *Proceedings of the 13th International Symposium on High-Performance Distributed Computing (HPDC'04)*. 172--181.

[213] Bin Xu, Jianzhong Huang, Qiang Cao, and Xiao Qin. 2019. TEA: A traffic-efficient erasure-coded archival scheme for in-memory stores. In *Proceedings of the 48th International Conference on Parallel Processing (ICPP '19)*. 24--34.

[214] Bin Xu, Jianzhong Huang, Qiang Cao, Xiao Qin, and Ping Xie. 2021. F-Write: Fast RDMA-supported writes in erasure-coded in-memory clusters. In *Proceedings of the 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS '21)*. 817--826.

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

[216] Liangliang Xu, Min Lyu, Zhipeng Li, Cheng Li, and Yinlong Xu. 2022. A data layout and fast failure recovery scheme for distributed storage systems with mixed erasure codes. *IEEE Transactions on Computers* 71, 8 (2022), 1740--1754.

[217] Silei Xu, Runhui Li, Patrick P. C. Lee, Yunfeng Zhu, Liping Xiang, Yinlong Xu, and John C. S. Lui. 2014. Single Disk Failure Recovery for X-Code-Based Parallel Storage Systems. *IEEE Trans. Comput.* 63, 4 (2014), 995--1007.

[218] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchi Zhang, Jian-Guang Lou, et al. 2018. Improving service availability of cloud systems by predicting disk error. In *Proceedings of 2018 USENIX Annual Technical Conference (USENIX ATC 18) (FAST'18)*. 481--494.

[219] Gala Yadgar, Eitan Yaakobi, and Assaf Schuster. 2015. Write once, get 50% free: saving SSD erase costs using WOM codes. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST '15)*. 257--271.

[220] Guofeng Yang, Huangzhen Xue, Yunfei Gu, Chentao Wu, Jie Li, Minyi Guo, Shiyi Li, Xin Xie, Yuanyuan Dong, and Yafei Zhao. 2022. XHR-Code: An efficient wide stripe erasure code to reduce cross-rack overhead in cloud storage systems. In *Proceeding of the 2022 41st International Symposium on Reliable Distributed Systems (SRDS '22)*. 273--283.

[221] Juncheng Yang, Anirudh Sabnis, Daniel S. Berger, K. V. Rashmi, and Ramesh K. Sitaraman. 2022. C2DN: How to harness erasure codes at the edge for fffficient content delivery. In *Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI'22)*. 1159--1177.

[222] Juncheng Yang, Yao Yue, and Rashmi Vinayak. 2021. Segcache: A memory-efficient and scalable in-memory key-value cache for small objects. In *Proceedings of the 18th Symposium on Networked Systems Design and Implementation (NSDI '21)*. 503--518.

[223] Qiaori Yao, Yuchong Hu, Liangfeng Cheng, Patrick P. C. Lee, Dan Feng, Weichun Wang, and Wei Chen. 2021. StripeMerge: Efficient wide-stripe generation for large-scale erasure-coded storage. In *Proceedings of the 2021 IEEE*

41st International Conference on Distributed Computing Systems (ICDCS '21). 483--493.

[224] Qiaori Yao, Yuchong Hu, Xinyuan Tu, Patrick P. C. Lee, Dan Feng, Xia Zhu, Xiaoyang Zhang, Zhen Yao, and Wenjia Wei. 2022. PivotRepair: Fast pipelined repair for erasure-coded hot storage. In *Proceedings of 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS '22)*. 614--624.

[225] Matt M. T. Yiu, Helen H. W. Chan, and Patrick P. C. Lee. 2017. Erasure coding for small objects in in-memory KV storage. In *Proceedings of the 10th ACM International Systems and Storage Conference (SYSTOR '17)*. 14--26.

[226] Qi Yu, Lin Wang, Yuchong Hu, Yumeng Xu, Dan Feng, Jie Fu, Xia Zhu, Zhen Yao, and Wenjia Wei. 2023. Boosting multi-block repair in cloud storage systems with wide-stripe erasure coding. In *Proceeding of the 2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS '23)*. 279--289.

[227] Guangyan Zhang, Zican Huang, Xiaosong Ma, Songlin Yang, Zhufan Wang, and Weimin Zheng. 2018. RAID+: Deterministic and balanced data distribution for large disk enclosures. In *Proceedings of 16th USENIX Conference on File and Storage Technologies (FAST '18)*. 279--294.

[228] Guangyan Zhang, Keqin Li, Jingzhe Wang, and Weimin Zheng. 2015. Accelerate RDP RAID-6 Scaling by Reducing Disk I/Os and XOR Operations. *IEEE Trans. Comput.* 64, 1 (2015), 32--44.

[229] Guangyan Zhang, Jiwu Shu, Wei Xue, and Weimin Zheng. 2007. SLAS: An efficient approach to scaling round-robin striped volumes. *ACM Transactions on Storage* 3, 1 (2007), 3--.

[230] Guangyan Zhang, Jiwu Shu, Wei Xue, and Weimin Zheng. 2010. ALV: A New Data Redistribution Approach to RAID-5 Scaling. *IEEE Trans. Comput.* 59, 3 (2010), 345--357.

[231] Guangyan Zhang, Weimin Zheng, and Keqin Li. 2014. Rethinking RAID-5 Data Layout for Better Scalability. *IEEE Trans. Comput.* 63, 11 (2014), 2816--2828.

[232] Mi Zhang, Shujie Han, and Patrick P. C. Lee. 2019. SimEDC: A Simulator for the Reliability Analysis of Erasure-Coded Data Centers. *IEEE Transactions on Parallel and Distributed Systems* 30, 12 (2019), 2836--2848.

[233] Mi Zhang, Qihan Kang, and Patrick P. C. Lee. 2023. Minimizing network and storage costs for consensus with flexible erasure coding. In *Proceedings of the 52nd International Conference on Parallel Processing (ICPP '23)*. 41--50.

[234] Shenglin Zhang, Ying Liu, Weibin Meng, Zhiling Luo, Jiahao Bu, Sen Yang, Peixian Liang, Dan Pei, Jun Xu, Yuzhi Zhang, et al. 2018. Prefix: Switch failure prediction in datacenter networks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 2, 1 (2018), 1--29.

[235] Yongzhe Zhang, Chentao Wu, Jie Li, and Minyi Guo. 2015. TIP-Code: A Three Independent Parity Code to Tolerate Triple Disk Failures with Optimal Update Complextiy. In *Proceedings of the 45th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'15)*. 136--147.

[236] Kai Zhao, Wenzhe Zhao, Hongbin Sun, Tong Zhang, Xiaodong Zhang, and Nanning Zheng. 2013. LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST '13)*. 243--256.

[237] Weimin Zheng and Guangyan Zhang. 2011. FastScale: Accelerate RAID Scaling by Minimizing Data Migration. In *Proceedings of the 9th USENIX Conference on File and Storage Technologies (FAST '11)*.

[238] Hai Zhou, Dan Feng, and Yuchong Hu. 2021. Multi-level forwarding and scheduling repair technique in heterogeneous network for erasure-coded clusters. In *Proceedings of the 50th International Conference on Parallel Processing (ICPP '21)*. 1--14.

[239] Hai Zhou, Dan Feng, and Yuchong Hu. 2023. MDTUpdate: A multi-block double tree update technique in heterogeneous erasure-coded clusters. *IEEE Transactions on Computers* 72, 10 (2023), 2808--2821.

[240] Panping Zhou, Jianzhong Huang, Xiao Qin, and Changsheng Xie. 2019. PaRS: A popularity-aware redundancy scheme for in-memory stores. *IEEE Transactions on Computers* 68, 4 (2019), 556--569.

[241] Tianli Zhou and Chao Tian. 2019. Fast erasure coding for data storage: A comprehensive study of the acceleration techniques. In *Proceedings of 17th USENIX Conference on File and Storage Technologies (FAST '19)*. 317--329.

[242] Yang Zhou, Hassan MG Wassel, Sihang Liu, Jiaqi Gao, James Mickens, Minlan Yu, Chris Kennelly, Paul Turner, David E. Culler, Henry M. Levy, et al. 2022. Carbink: Fault-tolerant far memory. In *Proceedings of 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*. 55--71.

[243] Yunfeng Zhu, Patrick P.C. Lee, Yinlong Xu, Yuchong Hu, and Liping Xiang. 2014. On the Speedup of Recovery in Large-Scale Erasure-Coded Storage Systems. *IEEE Transactions on Parallel and Distributed Systems* 25, 7 (2014), 1830--1840.

Received Nov. 29, 2023; revised Sep. 18, 2024; accepted Oct. 10, 2024