THE UNIVERSITY OF CHICAGO

SCALABLE MULTI-LEVEL ERASURE CODING IN HIERARCHICAL DATA CENTERS

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

DEPARTMENT OF COMPUTER SCIENCE

BY
MENG WANG

CHICAGO, ILLINOIS
2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Storage systems require data redundancy strategies to protect data from drive failures. As the sheer size, scale, complexity, and layering of distributed mass-capacity storage continue their unabated growth, both the frequency of drive failures and the time to rebuild a failed drive are growing. Therefore, data durability approaches must continue to evolve.

Existing storage systems mostly adopt single-level erasure coding (SLEC) to protect data, either using network-only SLEC or local-only SLEC. However, both SLEC approaches have limitations, as network-only SLEC introduces heavy network traffic overhead, and local-only SLEC cannot tolerate rack failures. Moreover, the typical ways to improve the durability of SLEC introduce either higher capacity overhead or slower encoding throughput.

Accordingly, we herein propose to study multi-level erasure coding (MLEC), which is a hybrid between the two existing SLEC approaches. We envision that MLEC will provide a better tradeoff between durability, capacity overhead, encoding/decoding CPU overhead, repair overhead (network traffic and rebuild IO), and update overhead.

In this project, we propose to (1). study the detailed design of MLEC, (2). build a mathematical model to compute durability for both SLEC and MLEC, (3). develop a simulator to simulate the failures and repairs of drives in large hierarchical data centers with different erasure approaches, (4). evaluate and compare the durability and overheads of SLEC, MLEC, and other erasure approaches under independent failures and correlated failure bursts.

CHAPTER 1

THESIS STATEMENT

Erasure coding approaches trade capacity overhead and encoding computation overhead for durability. We propose multi-level erasure coding (MLEC), which we envision to have a better tradeoff between durability and overheads than existing single-level erasure coding (SLEC) approaches. We study the design choices of MLEC. We prove the benefits of MLEC by building a Markov Chain model and developing a Monte Carlo simulator to measure the durability and overheads of SLEC, MLEC, and other erasure approaches. We evaluate MLEC and SLEC under both independent failures and correlated failure bursts to show the benefits of MLEC.

CHAPTER 2

INTRODUCTION

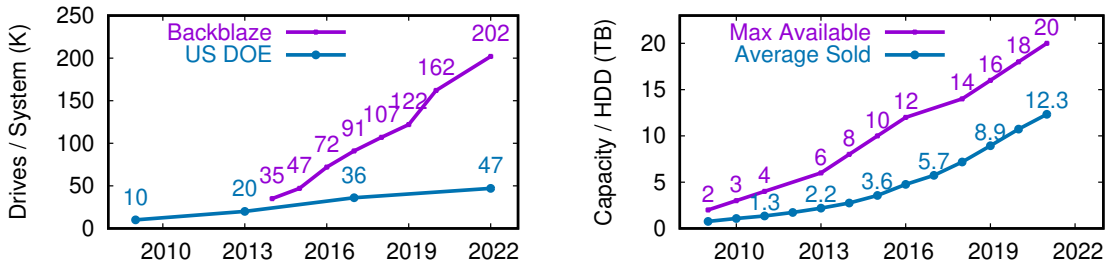
Storage systems need data redundancy mechanisms to protect data from the inevitable failures in data centers. Ever since computer data has been stored, two trends have been constant: the continued growth of the total number of storage devices used in a data center system (as shown in Figure 2.1(a)) and the complementary growth of the amount of capacity per individual device (Figure 2.1(b)). Despite these changes, the fundamental contract between storage and user, that a piece of data stored can be retrieved at an arbitrary future date without corruption, remains the same.

However, as the sheer size, scale, complexity, and layering of distributed mass-capacity storage continue their unabated growth, redundancy strategies must continue to evolve. The fundamental problem, summarized in Table 2.1, is that both the frequency of drive failure and the time to rebuild a failed drive are growing: a new durability approach is needed.

Due to the high capacity overhead of replication, most systems today use parity [34]. To improve rebuild times at scale, parity is typically declustered [8, 9, 20, 31, 41] such that every *stripe* of data, along with its associated parity, is distributed across a random (typically algorithmically pseudorandom) set of devices such that the number of devices participating in the rebuild is higher than would be the case in classic RAID systems. Finally, RAID systems which can tolerate the loss of up to two device failures (i.e. RAID6 [6]) have largely been replaced by more flexible *erasure coding* systems which can tolerate additional failures.

Trends		**Implications**	
HDDs / system	$\Uparrow$	*Failures / day / system*	$\Uparrow$
Capacity / HDD	$\Uparrow$	*Rebuild work / failure*	$\Uparrow$
Failure rate / HDD	$\Longleftrightarrow$	*Feasibility of*	
Performance / HDD	$\Longleftrightarrow$	*Existing Approaches*	$\Downarrow$

Table 2.1: **Trends and Implications.** *Horizontal arrows do not necessarily indicate no growth in absolute numbers but rather that any observed change is effectively flat relative to the others.*



(a) Drives per System

(b) Capacity per Drive

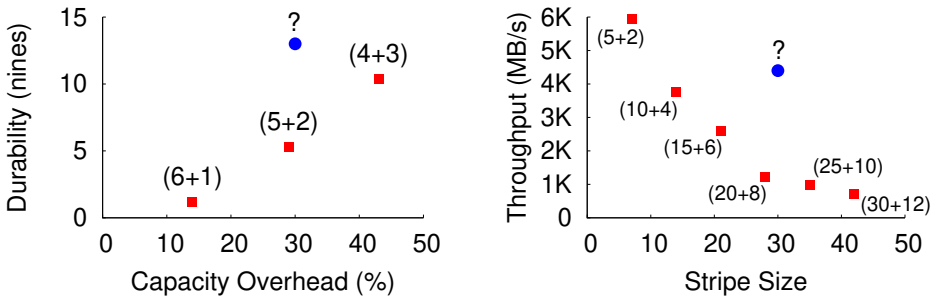
Figure 2.1: **Storage scaling over the years.** *The figures show (a) the increasing number of drives managed in Backblaze and US DOE laboratories and (b) per-drive capacity over the last 15 years.*

Thus far, most of the innovations in RAID and erasure coding have been in *single-level* protection strategies [13, 16, 17, 25, 27, 29, 35, 36, 37, 48, 49]. In these systems, a stripe of data is split into k *data chunks*, p *parity chunks* are computed, and these $(k+p)$ chunks are stored in different storage devices. When a device holding one of these chunks fails, the lost chunk can be restored from computation on the surviving chunks. Any $(k+p)$ erasure can tolerate up to p device failures without data loss.

There are several primary metrics by which durability mechanisms are evaluated: durability, capacity overhead, encoding/decoding CPU overhead, repair overhead (network traffic and rebuild IO), and update overhead. Problematically, the typical ways to improve durability increase one, or several of the overheads. As storage systems continue to scale in width and individual device capacity, existing erasure strategies that attempt to balance these metrics will hit a fundamental limitation.

To illustrate this problem, we quickly evaluate the tradeoffs among three metrics mentioned above: durability, capacity overhead, and encoding computational throughput. To increase durability, one can increase the ratio of parity to data; as shown in Figure 2.2(a), a $(5+2)$ coding ($\sim$29% capacity overhead) will deliver higher durability than a $(6+1)$ coding ($\sim$14% capacity overhead) system but requires higher capacity overhead.

Alternatively, durability can be increased by adding more parity without increasing capacity overhead if the ratio between data and parity is not changed. For example, $(5+2)$, $(10+4)$,



(a) Durability vs. Capacity Overhead (b) Throughput vs. Stripe Size

Figure 2.2: **Tradeoffs.** *The figures show the relationships between (a) capacity overhead/parity size and durability and (b) storage array width and the CPU overhead/parity computation throughput given a constant parity space overhead.*

and $(20+8)$ all have the same capacity overhead (*i.e.* 29%) but each provides respectively higher durability. Unfortunately, this increased durability comes with a steep price: the higher the value of k, the slower the computational throughput to perform the required erasure calculations. This can be seen in Figure 2.2(b), which shows throughput drops by $\sim 5\times$ from a $(10+4)$ stripe to a $(30+12)$ stripe.

Conversations with our industry partners reveal growing anxiety about mass-capacity data storage. As the Internet of Things produces an ever growing number of data streams, machine learning has emerged as a practical mechanism by which to extract actionable and monetizable insights from these ever larger data sets. However, economically efficient storage of this data with an acceptable risk of loss is quickly growing difficult. On multiple occasions, we have heard from our industry partners that their customers are facing increasingly challenging and disappointing victimization decisions as they are reluctantly forced to discard data that they can no longer afford to store.

These trends and these tradeoffs reveal the need for the next evolution of durability mechanisms as existing SLEC systems are quickly losing their ability to sufficiently protect data with reasonable capacity and CPU overheads. Thus, we want to address the following research questions: *Is there an erasure coding strategy that can deliver better durability at similar or lower overheads (as illustrated respectively by the "?" marks in Figures 2.2(a)) and 2.2(b))?*

To address this, we introduce a new approach of data durability: *multi-level erasure coding (MLEC)*.

First, we describe the MLEC architecture as a hybrid approach between two existing SLEC approaches: *local-only SLEC* as is used in HPC file systems [12, 18, 30], and *network-only SLEC* as is used in typical cloud systems [7, 15, 38, 39] and at least one HPC file system [47]. We will study the design details of MLEC, including data encoding, update and repair.

Second, we will build a mathematical model to compute the durability for different erasure coding approaches.

Third, we propose to develop a simulator to simulate the behaviors of different erasure approaches under drive failures. We will use the simulation results to validate our mathematical model.

Finally, we will evaluate and compare MLEC with other erasure approaches. We will use the Intel tool ISA-L [3] to measure the CPU throughput of encoding computations, and use the simulator and math model to measure other metrics, including durability, repair cost and update cost. We will evaluate these metrics against both independent disk failures and correlated failure bursts. We will analyze the tradeoffs between all these metrics. We envision that this combination of model, simulator, and empirical measurements will give us results showing that MLEC systems can provide more data durability at comparable overheads than existing SLEC systems.

CHAPTER 3

BACKGROUND

Here we briefly introduce the existing data redundancy approaches and their limitations.

3.1 Replication

The most straightforward way to protect data durability is replication [40], which makes multiple copies of the same data and distributes them across different disks. When a disk fails, its data can be recovered by reading from the copies in other disks. However, the price is the high storage overhead. For example, the commonly used 3-way replication requires 3x storage space as the original data.

3.2 RAID and Erasure Coding

Since replication introduces high storage overhead, nowadays many systems use parities to protect their data [34]. For example, the conventional RAID5 [5] systems use one parity per stripe and can tolerate any single device failure, and RAID6 [6] systems can tolerate up to 2 device failures by using 2 parities. In order to be more flexible and tolerate additional failures, many storage systems have started to use *erasure coding* (EC) [46].

In EC systems, a stripe of data is split into k *data chunks*, based on which p *parity chunks* are computed. These $(k+p)$ chunks are distributed into different storage devices. When one device fails, the lost chunks can be reconstructed from computation on the surviving chunks. Any $(k+p)$ erasure can tolerate up to p device failures without data loss. The conventional RAID5 systems utilizes $(k+1)$ erasure and RAID6 systems uses $(k+2)$ erasure.

3.3 Chunk Placement: Clustered vs. Declustered EC

When distributing chunks into storage devices, there are two typical kinds of data placement schemes: clustered and declustered.

In order to better illustrate the difference between clustered and declustered erasure, let's start with an example. Let's assume we want to do $(9+1)$ erasure among 100 drives.

In clustered erasure, we divide the 100 drives into 10 groups, each containing $9+1=10$ drives. Every stripe is assigned to a specific group, and the $(9+1)$ chunks are distributed among the 10 drives. A stripe either has no chunk in a group, or has all the chunks residing in the group. When a drive fails, we read from 9 surviving drives to reconstruct the lost data and write to a new spare drive. Only 10 drives participate in the rebuild, and the rebuild rate is bottlenecked by the single drive's read/write rate.

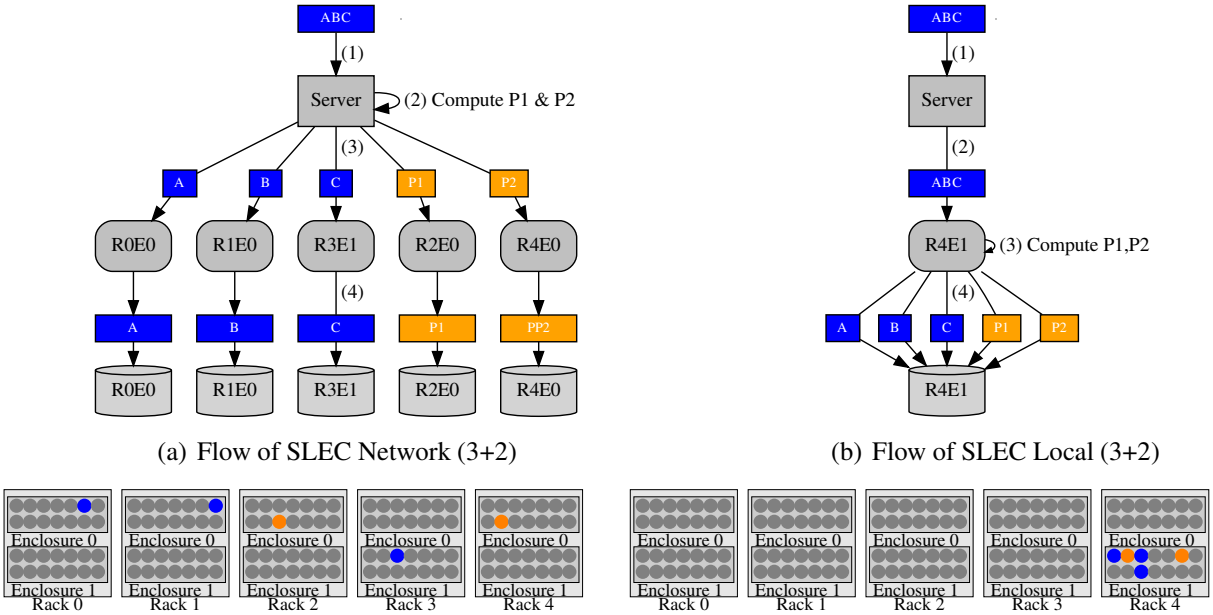
When using declustered erasure [2, 10, 19], every stripe is distributed across 10 pseudorandom drives. Each drive has some spare space where no data nor parity is stored. When a drive fails, it has a huge number of lost chunks on it. For each lost chunk, 9 pseudorandom surviving drives are read to reconstruct it, and the reconstructed chunk is written to the spare space on one pseudorandom drive. Since there are a huge number of lost chunks, the total build work is spread across all 99 surviving drives, and therefore the rebuild rate is much faster than that of clustered erasure.

The price for the faster rebuild is the less tolerance against concurrent failures. For example, if drive 1 and drive 99 fail simultaneously, in clustered erasure, the system is recoverable since the two failed drives belong to two different drive groups. In declustered erasure, however, due to its pseudorandom placement scheme, some stripes have chunks on both failed drives, and these 2-chunk-failure stripes can no longer be restored using $(9+1)$ erasure.

3.4 Local vs. Network SLEC

Many distributed storage systems today use declustered erasure and are built with similar hardware. Clients send and receive data from servers which, in turn, send and receive data between themselves and distributed storage *enclosures*. Storage enclosures are typically between four and five rack units high and contain around 100 drives. These enclosures contain *controllers* which either act merely as a path to the individual devices or add a degree of virtualization.

There are two existing single-level erasure coding (SLEC) approaches: *network-only SLEC* and local-only SLEC. To describe them, we imagine a data center that has five racks of storage, each holding two enclosures, each holding fourteen storage devices.



(a) Flow of SLEC Network (3+2)

(b) Flow of SLEC Local (3+2)

(c) Resulting Layout of SLEC Network (3+2)

(d) Resulting Layout of SLEC Local (3+2)

Figure 3.1: **Logical Erasure Workflows and Physical Layouts of SLEC.** *The upper figures show the logical flow and parity generation when new data is stored. Rounded rectangles represent the controllers within each enclosure, and cylinders represent their drives. The lower figures show the resulting physical layout in an exemplaryy data center. All figures use colors to indicate the type of data being stored: blue for the original user data, and orange for the parity computed from that data. For the purpose of elucidation, not all elements are shown in each figure. For example, servers are not shown in the physical layouts, and not all enclosures are shown in the logical flows. Each is configured such that all have the same capacity overhead (40%).*

As exemplified in Figure 3.1(a), a *network-only SLEC* server splits received data into three

data chunks, computes two parity chunks, and then distributes those chunks across enclosures by sending them to the controllers, which forward the data to the requested devices [7, 15, 38, 39]. Because racks and enclosures are well-known failure domains, typical network-only systems ensure a layout such that no stripe has more than p chunks per rack or enclosure.

In Figure 3.1(c), we show a typical layout in which each chunk is stored in a different rack and no one rack has more than one chunk. In this way, the system minimizes the number of chunks lost in any one stripe when a rack fails. Note that we assume that all approaches use declustered erasure such that the placement shown in the figures is only for one particular stripe of data. Other stripes will have different layouts (i.e. will be stored on different devices). The enclosures used in network-only architectures are often referred to as *JBOD* (just a bunch of disks).

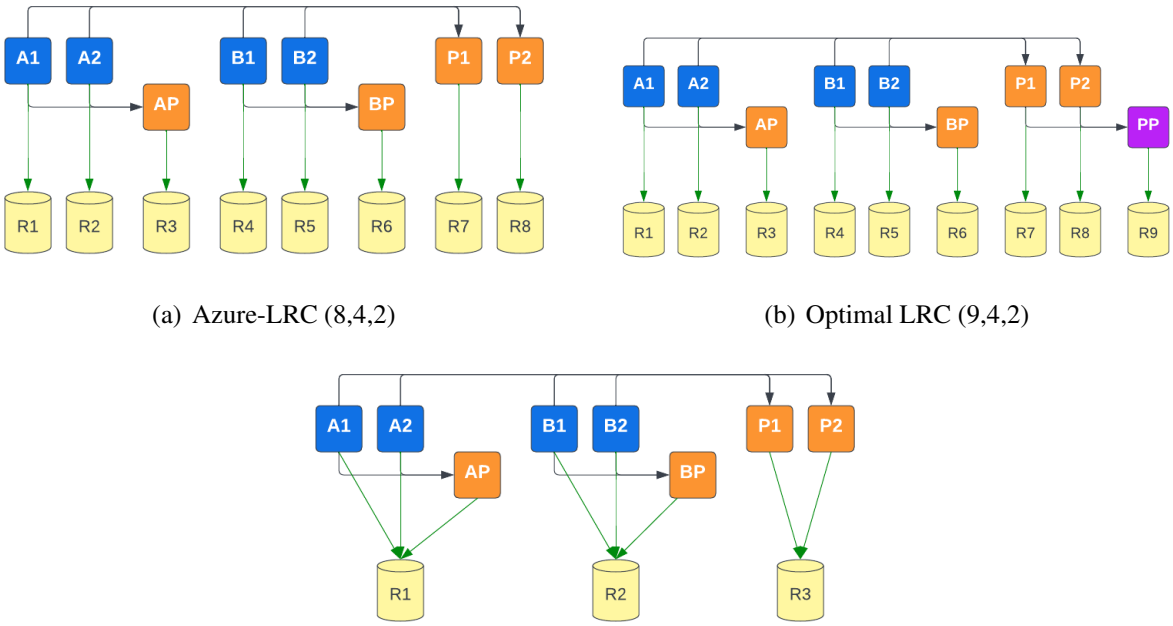
The second SLEC approach, *local-only SLEC* uses *RBOD* (reliable bunch of disks) instead of JBOD [12, 18, 30]. An RBOD, which is typically created using erasure firmware running on the controller, appears to the upper-level system as a very large drive. However, internally it is protecting data using erasure. As shown in Figures 3.1(b) and 3.1(d), a client sends data to a server, and that server passes the data directly to an RBOD controller, which then splits the data, computes the parity, and stores the five resulting chunks onto five of its drives.

Note that the work performed is fundamentally the same in both SLEC approaches with only two small differences. One is whether a server or a controller does the splitting, computing, and distributing, and the second is whether the distribution is done across enclosures or within a single one.

Also note that local-only SLEC does not protect data from rack or enclosure failures. Network-only SLEC does tolerate rack failures, but the repair introduces heavy network traffic.

3.5 Locally Repairable Coding

In $(k+p)$ erasure coding, repairing any lost chunk requires reading k surviving chunks from other disks. When k is large, the repair becomes IO-intensive and computation-intensive. Locally Re-



(a) Azure-LRC (8,4,2)

(b) Optimal LRC (9,4,2)

(c) Combined Locality (8,4,2,3)

Figure 3.2: **Locally repairable coding (LRC).** *The figures illustrate the workflows of different LRC approaches including (a) Azure-LRC (b) optimal-LRC (c) Combined Locality.*

pairable Coding (LRC) deals with this problem by adding local parities.

For example, in (n,k,r) Azure-LRC [22], k data chunks are divided into $\lceil k/r \rceil$ local groups. Each local group contains r data chunks, and one local parity is computed. Meanwhile, it encodes $n-k-\lceil k/r \rceil$ global parity chunks using all data chunks. This results in n chunks in total per stripe. When a data chunk or local parity is lost, it can be reconstructed by reading only r surviving chunks in the same local group. Figure 3.2(a) shows an example of LRC $(8,4,2)$, which has 4 data chunks, 2 local parities, and 2 global parities.

Note that the repair of global parities in Azure LRC still requires reading k surviving chunks. Such LRCs are referred to as *data-LRCs* [26]. On the other hand, *full-LRCs* can repair any chunk (including the global parity chunk) using r surviving chunks.

Optimal-LRC [43] is a typical full-LRC, where all data chunks and global parity chunks are divided into local groups of size r, and each local group has one local parity. For example, Figure 3.2(b) shows optimal-LRC (9,4,2).

Note that Azure-LRC distributes each chunk of a stripe into a different rack. By doing this, it can tolerate rack failures. However, the repair of any lost chunk will introduce cross-rack network traffic.

Azure-LRC+1 [26], in the evaluation, first tried to place a local group in a rack to reduce cross-rack repair network traffic.

Combined locality (CL) [21] first systematically explores how to deploy LRC in a rack-hierarchical structure in order to reduce network traffic. In CL (n,k,r,z), they added a new parameter z to denote how many stripes to place a stripe. For example, Figure 3.2(c) shows CL (8,4,2,3) which distributes the 8 chunks into 3 racks. In this example, repairing data chunks or local parities requires 0 network traffic. Repairing the global parity chunk, however, requires 4 chunks of cross-rack network traffic cost.

3.6 Nested RAID

Nested RAID, also known as hierarchical RAID, or 2D RAID, combines two RAID levels to increase durability [11]. For example, RAID 55 performs RAID 5 at the top level, and another RAID 5 at the bottom level. Many works have been done to analyze the reliability and performance of nested RAID [11, 28, 33, 42, 44, 45]. However, some critical questions remain to be answered:

1. All the published works consider deploying nested RAID in a local disk group. However, none has considered a rack-hierarchical data center, where rack failures can happen, and cross-rack network bandwidth is much more expensive and constrained than inner-rack bandwidth.

2. Among the published works, only OI-RAID [28] considered declustered parity. They discussed how to apply declustered parity in RAID 55. However, they didn't perform a detailed analysis of the durability of declustered nested RAID.

3. As mentioned earlier, the increased durability in erasure coding always introduces one or

more overhead costs, including capacity overhead, CPU throughput cost, update cost, etc. However, no published work has compared the durability of nested RAID with normal RAID under similar overheads or vice versa.

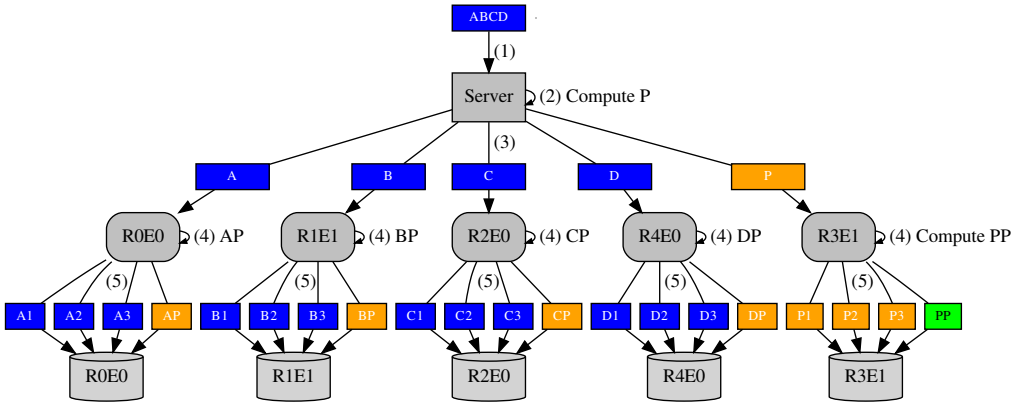
4. In massive storage systems, correlated failures can happen [14, 32]. However, no published work has analyzed the reliability of nested RAID against correlated failure bursts.

Therefore, we will try to answer these questions in our research.

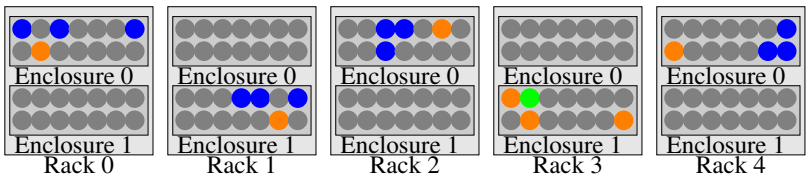
CHAPTER 4

MLEC DESIGN

4.1 Architecture



(a) Flow of MLEC (4+1)/(3+1)



(b) Resulting Layout of MLEC (4+1)/(3+1)

Figure 4.1: **MLEC Architecture.** *The upper figures show the logical flow and parity generation when new data is stored. The lower figures show the resulting physical layout in an exemplary data center.*

We propose multi-level erasure coding (MLEC) architecture (shown in Figure 4.1(a)), which is a combination of the two SLEC approaches. In MLEC, a server receiving data splits that data into chunks, computes parity chunks, and then distributes those, in a rack-aware layout, across enclosures. Each enclosure, an RBOD, takes the data it receives, further splits it, computes additional parity, and then stores those *subchunks* across its internal drives.

Note that an RBOD receiving a chunk from the server might be receiving a data chunk, or it might be receiving a parity chunk. An RBOD receiving a data chunk will split it into data subchunks and add parity subchunks, whereas an RBOD receiving a parity chunk will split it into parity subchunks and add *double-parity* subchunks. Note that an RBOD is unaware whether the

chunk it receives is data or parity, but we describe this nonetheless to illustrate the precise flow of data, parity, and double-parity through the system. We use $(k+p)$ to describe SLEC systems and $(k_{\mathrm{n}}+p_{\mathrm{n}})/(k_{\ell}+p_{\ell})$ to describe MLEC systems where the n and l stand respectively for "network" and "local".

A $(k_n+p_n)/(k_l+p_l)$ MLEC can tolerate any $(p_l+1)*p_n+p_l$ arbitrary failures. This is because in order to cause an MLEC system to fail, the storage system needs at least p_n+1 rack failures, and each rack failure requires at least p_l+1 disk failures in that rack. Thus at least $(p_l+1)*(p_n+1)$ disk failures are required to cause the MLEC system to fail.

While there are at least three production MLEC systems today: LANL MarFS [23], Seagate CORTX [4], and NetApp StorageGrid [1], none have published any detailed analysis of MLEC design. Therefore in our research, we will introduce the current designs in production and discuss possible design improvements.

4.2 Chunk Placement

When implementing local erasure, we can choose to use clustered erasure or declustered erasure.

As is shown in Figure 4.2, clustered erasure uses deterministic chunk placement, which results in more concurrent failure tolerance but slower repair speed. On the other hand, declustered uses random chunk placement, which leads to faster repair speed but less concurrent failure tolerance. Existing production MLEC systems like LANL MarFS and Seagate CORTX both use declustered erasure locally.

Single-overlap Declustered Parity (SODP) [24] is an extension to declustered erasure, which increases its concurrent failure tolerance by reducing the randomness in its placement. For example, suppose we are doing $4+1$ among 50 disks. When using declustered parity, there are $C(50,5)=2118760$ possible ways to place a stripe of 5 chunks. SODP reduces the randomness by using only 120 possible placements.

SODP is proven to provide a better tradeoff between repair rate and concurrent failure tolerance

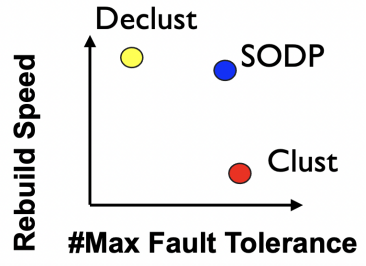


Figure 4.2: **Tradeoff for Different Chunk Placement Policies in local-only SLEC.** *The figure illustrates the tradeoff between repair speed and fault tolerance for different chunk placement policies in local-only SLEC.*

in local-only SLEC [24]. However, it's unknown how it should be applied to MLEC *(should we apply SODP in network-level erasure or local-level erasure or both?)* and how it performs in MLEC. Therefore, we will analyze and evaluate it in our research.

4.3 Encoding

4.3.1 Where Does Encoding Happen

In MLEC, we need to perform two levels of encoding, and there are two ways to arrange the encoding work:

1. We can let the top-level server perform top-level encoding, and let the bottom local controller perform local-level encoding. The benefit is the easy deployment. Since many production storage systems already use RBODs which contain a local encoding controller, we can build MLEC on top of these RBODs and thus take advantage of these controllers.

2. We can also let the top-level server perform both top-level and bottom-level encoding. The benefit is that we may make use of the data locality in the cache and thus speed up the overall encoding throughput.

Currently, both LANL MarFS and Seagate CORTX use the first method. We will want to evaluate and compare it with the second method in our future research.

(a) Encode with large chunks at the network level (b) Encode with small chunks at the network level

Figure 4.3: **Encoding design.** *(a) Encode with large chunks in network erasure (b) Encode with small chunks in network erasure*

4.3.2 Top-level Chunk Size

Both Seagate CORTX and LANL MarFS treat the bottom-level disk group as a "giant disk". Thus, the top-level server has no knowledge of the bottom-level erasure. When a stripe of data arrives, it's first divided into large chunks, and top-level erasure is performed. The large chunk is further divided into small chunks, and local-level erasure is performed. Figure 4.3(a) shows an example for $(4+1)/(4+1)$.

Encoding large chunks in network erasure may have several potential disadvantages:

- When we update a single small device chunk A1, we need to update the corresponding network parity. If we use large chunks in network-level EC, we will need to transfer the large chunk $A_{new} - A_{old}$ cross-rackly, which is unnecessary.

- Similarly, when A1 and A2 are lost, they must be repaired via network-level erasure. In this case, we need to treat the large chunk A as lost and read B, C, D, P from other racks to

compute A. However, this involves unnecessary work because A3 and A4 are still alive and don't require reconstruction.

Actually, it's not unnecessary to encode large chunks in network-level erasure. As is shown in figure 4.3(b), before doing network erasure coding, we can first divide each large chunk, e.g., A, into small chunks A_1, A_2, A_3, A_4. Then we perform network level erasure coding on A_1, B_1, C_1, D_1 and get P_1. Similarly we can get P_2, P_3, P_4.

When using the same encoding matrix, the parity values of small chunk erasure coding [P1, P2, P3, P4] are the same as the parity value of large chunk erasure coding P. Thus, using small chunks for network erasure encoding doesn't affect the encoding result.

Encoding with small chunk size is advantageous because:

- When we want to update the network parity for a single small device chunk A_1, we only need to transfer the small chunk $A_{1,new} - A_{1,old}$
- Similarly, when A_1 and A_2 are lost, we can reconstruct them directly with less network traffic and computation overhead.

4.4 Update

When we update a chunk A_1, we need to also update the corresponding local parity A_P, network parity P_1, and double parity P_P.

In order to perform the update, we compute the delta $A_{1,new} - A_{1,old}$. Then the local parity A_P can be updated as $A_{P,new} = A_{P,old} + \alpha' \cdot (A_{1,new} - A_{1,old})$, where α' is the corresponding encoding coefficient in local erasure. This method doesn't need to read A_2, A_3, A_4 and the computation is simple.

Similarly, P_1 can be updated as $P_{1,new} = P_{1,old} + \alpha \cdot (A_{1,new} - A_{1,old})$, where α is the corresponding encoding coefficient in network erasure. The update of network parity requires one chunk of cross-rack traffic.

And P_P can be updated as $P_{P,new} = P_{P,old} + \alpha' \cdot (P_{1,new} - P_{1,old})$

Therefore, each data chunk update only requires 1 chunk of cross-rack network traffic.

4.5 Repair

When no more than p_l disks fail in a disk group, the repair is straightforward: each lost chunk can be repaired by reading k_l surviving chunks in the same disk group and performing local erasure without any network traffic.

When more than p_l disks fail in the group, we need to perform network repair. And we will have several repair options.

Let's take MLEC $(4+1)/(3+1)$ in Figure 4.1(a) as an example. When only disk A_1 fails, it can be repaired locally and we say the affected stripes are in the *degraded state*. When there are two disk failures A_1, A_2 in the local disk group A, we need to repair them using network erasure.

4.5.1 If local does clustered erasure

If the local level deploys clustered erasure, then there are several options on how to perform network erasure:

1. Treat the whole disk group as failed, and reconstruct the entire disk group using network erasure.

2. Only repair the failed two disks. Repair all the failed stripes using network erasure.

3. Repair all failed stripes to degraded (i.e., only repair one chunk for each stripe). Later each degraded stripe can be repaired locally.

Method 1 will definitely require the most network traffic, and method 3 will require the least network traffic. However, it's unknown which method provides the most durability. Therefore, we will need to evaluate these 3 methods in the future.

4.5.2 If local does declustered erasure

If the local level adopts declustered erasure, the repair method options are similar to clustered erasure. However, note that when two disks A_1, A_2 fail, not all the stripes related to A_1, A_2 fail. Most affected stripes only have one chunk failed, either residing on A_1 or A_2. These stripes are in the degraded state and can be repaired locally. Only a few stripes have chunks both on A_1 and A_2. These stripes are critical stripes that require network-level repair. Once these critical stripes are repaired, the disk group can repair itself locally.

Thus, now we have 4 different repair choices:

1. Treat the whole disk group as failed, and reconstruct the entire disk group using network erasure.

2. Only repair the failed two disks. Repair all the affected stripes using network erasure.

3. Repair all critical stripes to healthy using network erasure. Other degraded stripes can be repaired locally.

4. Repair all critical stripes to degraded (i.e., only repair one chunk for each stripe). These repaired-to-degraded stripes, along with other affected stripes, can be repaired locally.

Currently, Seagate CORTX utilizes the 1st method as it's easy to implement and deploy. However, if we compare the repair network traffic for these methods, it should be method 1 $>$ method 2 $>$ method 3 $>$ method 4. Further analysis needs to be made to evaluate and compare the durability of these 4 repair options.

4.6 MLEC vs. LRC

Readers may be confused what's the difference between MLEC and LRC, as they both have local parities and network parities (in LRC they are called global parities.)

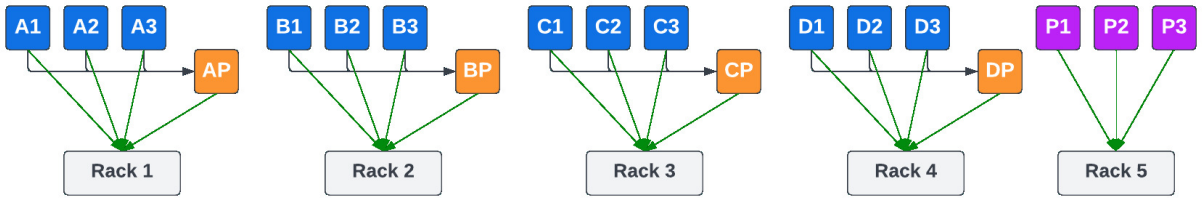


Figure 4.4: **Logical Flow of CL-LRC(19,12,3,5).** *The figure illustrates the logical flow of Combined Locality (19,12,3,5), which divides 12 data chunks into 4 groups of size 3. Each group computes one local parity. 3 global parities are computed. The total 19 chunks are distributed into 5 racks.*

The major difference is how the network (global) parities are computed.

For better illustration, let's still use $(4+1)/(3+1)$ in Figure 4.1(a) as an MLEC example. For the LRC example, let's consider CL-LRC (19, 12, 3, 5) as is shown in Figure 4.4, which has 1 local parity per 3 data chunks, and 3 global parities per 12 data chunks, with a total 19 chunks distributed into 5 racks.

In CL-LRC, the global parity P_1 is a linear combination of all the 12 data chunks, which requires 12 multiplications and 12 additions:

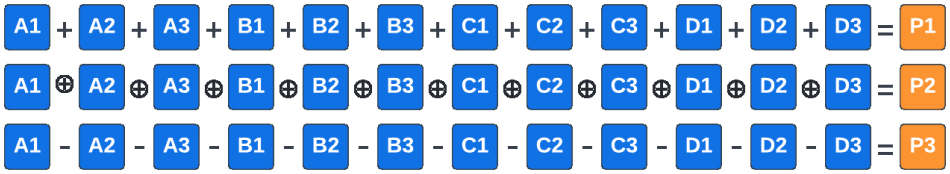
$$P_1 = \alpha_{1,1}A_1 + \alpha_{1,2}A_2 + \alpha_{1,3}A_3 + \beta_{1,1}B_1 + \beta_{1,2}B_2 + \beta_{1,3}B_3 + \gamma_{1,1}C_1 + \gamma_{1,2}C_2 + \gamma_{1,3}C_3 + \delta_{1,1}D_1 + \delta_{1,2}D_2 + \delta_{1,3}D_3$$

In MLEC, on the other hand, the network parity P_1 is a linear combination of 4 data chunks, which requires only 4 multiplications and 4 additions:

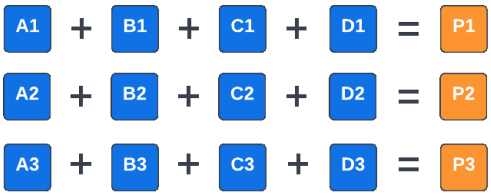
$$P_1 = \alpha A_1 + \beta B_1 + \gamma C_1 + \delta D_1$$

The detailed computation difference can be illustrated in Figure 4.5.

Since LRC uses a more complex encoding when computing global parities, it provides more durability against disk failures. As a tradeoff, it requires more computation overhead, more update



(a) CL-LRC(19,12,3,5) Global Parity Encoding



(b) MLEC (4+1)/(3+1) Network Parity Encoding

Figure 4.5: **Global(network) parity encoding for CL-LRC and MLEC.** *The figure illustrates the difference in network(global) parity encoding computation between CL-LRC(19,12,3,5) and MLEC (4+1)/(3+1). For simplicity, we don't show the encoding coefficients in the figure. Instead, we use different operation symbols to illustrate that different parities are encoded using different computations.*

cost (when LRC updates A_1, it needs to update A_P, P_1, P_2, P_3, P_4), more network repair computation overhead, network traffic, and rebuild IO. Therefore, we will need to compare the overheads of MLEC and LRC with comparable durability.

CHAPTER 5

MODELING DATA DURABILITY

We will model the data durability using Markov Chain, as shown in Figure 5.1.

Each circle represents a system state, and the number in the circle represents the number of failures in the system.

Suppose an erasure-coded system has S storage devices, and each storage devices fail at the respective constant rate of λ. Then in the system single device failures happen at the rate of $S\lambda$. When failures happen, the failed disk is repaired at a specific repair rate μ. It's possible for another disk to fail before the failed disks are repaired. The system can tolerate at most k failures. When there are $k+1$ failures, data has been lost (DL).

When building the model, one important thing is to quantify the repair rate μ. The repair rate depends on the system scale, the adopted erasure coding approach, and the deployed repair approach. Therefore, we will need to analyze the repair rate for different erasure coding approaches, including SLEC, MLEC, and LRC, and model their durability.

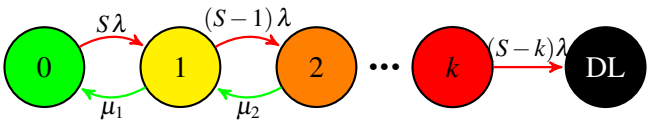


Figure 5.1: **Markov Chain of System States.** *An erasure-coded system has S storage devices. Each storage device fails at the respective constant rate of λ. When failures happen, the failed disk is repaired at a specific repair rate μ. The system can tolerate at most k failures. At $k+1$ failures, data has been lost (DL).*

CHAPTER 6

SIMULATOR DESIGN

We will build a Monte Carlo simulator to simulate how a storage cluster behaves against disk failures and rack failures under different erasure coding policies. We will run the simulator against both independent failures and correlated failure bursts. Based on simulation results, the simulator should be able to report system durability, which will be used to verify the math model. The simulator will also report other metrics, including network traffic, rebuild IO, etc., which will be used in the evaluation section to compare MLEC with SLEC and LRC.

The simulator should work in this flow:

- Given a storage system composed of X racks and Y disks per rack, we generate $X \cdot Y$ random disk failures, assuming disk failure follows a particular distribution.
- We simulate rebuild behaviors of different EC approaches to see if the system will fail in one year.
- We repeat the simulation M times, and get N system failures and M-N survivals. Then the probability of data loss is N/M.
- Thus the system durability, which is the probability of system survival, will be $1 - N/M$.

The durability results from the simulations can be used to verify the math model. We can also measure other metrics using the simulator.

CHAPTER 7

EVALUATION

We will evaluate and compare MLEC, SLEC, and LRC under independent failures as well as correlated failure bursts.

The CPU computation throughput will be measured using the Intel ISA-L tool [3].

Other metrics will be measured using the math model and the simulation results.

We will compare the overheads of different erasure approaches given similar durability.

CHAPTER 8

RESEARCH PROGRESS AND PLAN

Project	Progress	Descriptions
Background Literature Review	✓	we have completed the literature review on existing erasure coding approaches
MLEC Design	✓	we have come up with an MLEC architecture which combines the benefits of the two SLEC approaches, and we've proposed several design choices.
Durability Modeling	□	we plan to build a Markov Chain model to compute the durability of MLEC, SLEC, and LRC in Autumn'22
Simulator Development	□	we plan to implement a simulator to simulate the behaviors of MLEC, SLEC, and LRC in Winter'23
MLEC Design Choice Analysis	□	we plan to analyze different MLEC design choices with the help of the simulator in Spring'23
Evaluation	□	we plan to evaluate MLEC, SLEC, and LRC against both independent failures and correlated failure bursts in Summer'23
Dissertation	□	we plan to write the dissertation and finish the defense in Autumn'23

Table 8.1: **Researh Progress and Plan.**

REFERENCES

[1] Hierarchical Erasure Coding: Making Erasure Coding Usable. https://www.snia.org/sites/default/files/SNIA_Hierarchical_Erasure_Coding_Final.pdf.

[2] Declustered raid. https://www.ibm.com/support/knowledgecenter/en/SSYSP8_5.1.0/com.ibm.spectrum.scale.raid.v4r23.adm.doc/bl1adv_introdeclustered.htm.

[3] Intel Intelligent Storage Acceleration Library (Intel ISA-L). https://software.intel.com/en-us/storage/ISA-L.

[4] Lyve Rack - Open Object Storage, Simplified. https://www.seagate.com/products/storage/object-storage-solutions/lyve-drive-rack/.

[5] Raid 5. https://searchstorage.techtarget.com/definition/RAID-5-redundant-array-of-independent-disks.

[6] Raid 6. https://searchstorage.techtarget.com/definition/RAID-6-redundant-array-of-independent-disks.

[7] Abutalib Aghayev, Sage Weil, Michael Kuchnik, Mark Nelson, Gregory R. Ganger, and George Amvrosiadis. File systems unfit as distributed storage backends: lessons from 10 years of Ceph evolution. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, 2019.

[8] Guillermo A. Alvarez, Walter A. Burkhard, and Flaviu Cristian. Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering. In *Proceedings of the 24th Annual International Symposium on Computer Architecture (ISCA)*, 1997.

[9] Guillermo A. Alvarez, Walter A. Burkhard, Larry J. Stockmeyer, and Flaviu Cristian. Declustered disk array architectures with optimal and near-optimal parallelism. In *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA)*, 1998.

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

[11] Sung Hoon Baek, Bong Wan Kim, Eui Joung Joung, and Chong Won Park. Reliability and performance of hierarchical raid with multiple controllers. In *Proceedings of the twentieth annual ACM symposium on Principles of Distributed Computing*, pages 246–254, 2001.

[12] Peter Braam. The lustre storage architecture. *arXiv preprint arXiv:1903.01955*, 2019.

[13] Jeremy C. W. Chan, Qian Ding, Patrick P. C. Lee, and Helen H. W. Chan. Parity Logging with Reserved Space: Towards Efficient Updates and Recovery in Erasure-coded Clustered Storage. In *Proceedings of the 12th USENIX Symposium on File and Storage Technologies (FAST)*, 2014.

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

[15] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[16] James Lee Hafner. WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems. In *Proceedings of the 4th USENIX Symposium on File and Storage Technologies (FAST)*, 2005.

[17] James Lee Hafner, Veera Deenadhayalan, KK Rao Center, and John A. Tomlin. Matrix Methods for Lost Data Reconstruction in Erasure Codes. In *Proceedings of the 4th USENIX Symposium on File and Storage Technologies (FAST)*, 2005.

[18] Frank Herold, Sven Breuner, and Jan Heichler. An introduction to beegfs, 2014.

[19] Mark Holland and Garth Gibson. Parity declustering for continuous operation in redundant disk arrays. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 1992.

[20] Mark Holland and Garth A. Gibson. Parity declustering for continuous operation in redundant disk arrays. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1992.

[21] Yuchong Hu, Liangfeng Cheng, Qiaori Yao, Patrick PC Lee, Weichun Wang, and Wei Chen. Exploiting combined locality for {Wide-Stripe} erasure coding in distributed storage. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 233–248, 2021.

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

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

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

[25] Osama Khan, Randal Burns, James Plank, William Pierce, and Cheng Huang. Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads. In *Proceedings of the 10th USENIX Symposium on File and Storage Technologies (FAST)*, 2012.

[26] Oleg Kolosov, Gala Yadgar, Matan Liram, Itzhak Tamo, and Alexander Barg. On fault tolerance, locality, and optimality in locally repairable codes. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, pages 865–877, 2018.

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

[28] Yongkun Li, Neng Wang, Chengjin Tian, Si Wu, Yueming Zhang, and Yinlong Xu. A hierarchical raid architecture towards fast recovery and high reliability. *IEEE Transactions on Parallel and Distributed Systems*, 29(4):734–747, 2017.

[29] Umesh Maheshwari. StripeFinder: Erasure Coding of Small Objects Over Key-Value Storage Devices (An Uphill Battle). In *the 12th Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2020.

[30] Michael Moore, David Bonnie, Walt Ligon, Nicholas Mills, Becky Ligon, Mike Marshall, Elaine Quarles, and Sam Sampson. OrangeFS: Advancing PVFS. In *Proceedings of the 9th USENIX Symposium on File and Storage Technologies (FAST)*, 2011.

[31] Richard R. Muntz and John C. S. Lui. Performance analysis of disk arrays under failure. In *Proceedings of the 16th International Conference on Very Large Data Bases (VLDB)*, 1990.

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

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

[34] David Patterson, Garth Gibson, and Randy Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the 1988 ACM SIGMOD Conference on the Management of Data (SIGMOD)*, 1988.

[35] James S. Plank, Mario Blaum, and James L. Hafner. SD Codes: Erasure Codes Designed for How Storage Systems Really Fail. In *Proceedings of the 11th USENIX Symposium on File and Storage Technologies (FAST)*, 2013.

[36] KV Rashmi, Preetum Nakkiran, Jingyan Wang, Nihar B. Shah, and Kannan Ramchandran. Having Your Cake and Eating It Too: Jointly Optimal Erasure Codes for I/O, Storage and Network-bandwidth. In *Proceedings of the 13th USENIX Symposium on File and Storage Technologies (FAST)*, 2015.

[37] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. XORing Elephants: Novel Erasure Codes for Big Data. In *Proceedings of the 39th International Conference on Very Large Data Bases (VLDB)*, 2013.

[38] Omar SEFRAOUI, Mohammed AISSAOUI, and Mohsine ELEULDJ. OpenStack: toward an open-source solution for cloud computing. In *International Journal of Computers and Applications (IJCA)*, 2012.

[39] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *Proceedings of the 26th IEEE Symposium on Massive Storage Systems and Technologies (MSST)*, 2010.

[40] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, et al. The hadoop distributed file system. In *MSST*, volume 10, pages 1–10, 2010.

[41] Thomas J.E. Schwarz S.J., Jesse Steinberg, and Walter A. Burkhard. Permutation development data layout (PDDL). In *Proceedings of the 5th International Symposium on High Performance Computer Architecture (HPCA-5)*, 1999.

[42] Tae-Geon Song, Mehdi Pirahandeh, Cheong-Jin Ahn, and Deok-Hwan Kim. Gpu-accelerated high-performance encoding and decoding of hierarchical raid in virtual machines. *The Journal of Supercomputing*, 74(11):5865–5888, 2018.

[43] Itzhak Tamo and Alexander Barg. A family of optimal locally recoverable codes. *IEEE Transactions on Information Theory*, 60(8):4661–4676, 2014.

[44] Alexander Thomasian. Multi-level raid for very large disk arrays. *ACM SIGMETRICS Performance Evaluation Review*, 33(4):17–22, 2006.

[45] Alexander Thomasian and Yujie Tang. Performance, reliability, and performability aspects of hierarchical raid. In *2011 IEEE Sixth International Conference on Networking, Architecture, and Storage*, pages 92–101. IEEE, 2011.

[46] Hakim Weatherspoon and John D Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *International Workshop on Peer-to-Peer Systems*, pages 328–337. Springer, 2002.

[47] Brent Welch, Marc Unangst, Zainul Abbasi, Garth Gibson, Brian Mueller, Jason Small, Jim Zelenka, and Bin Zhou. Scalable Performance of the Panasas Parallel File System. In *Proceedings of the 6th USENIX Symposium on File and Storage Technologies (FAST)*, 2008.

[48] Mingyuan Xia, Mohit Saxena, Mario Blaum, and David A. Pease. A Tale of Two Erasure Codes in HDFS. In *Proceedings of the 13th USENIX Symposium on File and Storage Technologies (FAST)*, 2015.

[49] Guangyan Zhang, Zican Huang, Xiaosong Ma, Songlin Yang, Zhufan Wang, and Weimin Zheng. RAID+: Deterministic and Balanced Data Distribution for Large Disk Enclosures. In *Proceedings of the 16th USENIX Symposium on File and Storage Technologies (FAST)*, 2018.