Design Considerations and Analysis of Multi-Level Erasure Coding in Large-Scale Data Centers

Meng Wang
University of Chicago
Chicago, IL, USA
wangm12@uchicago.edu

Jiajun Mao
University of Chicago
Chicago, IL, USA
jiajunm@uchicago.edu

Rajdeep Rana
University of Chicago
Chicago, IL, USA
rajrana22@uchicago.edu

John Bent
Los Alamos National Laboratory
Los Alamos, NM, USA
johnbent@gmail.com

Serkay Olmez
Seagate Research
Longmont, CO, USA
serkay.olmez@seagate.com

Anjus George
Oak Ridge National Laboratory
Oak Ridge, TN, USA
georgea@ornl.gov

Garrett Wilson Ransom
Los Alamos National Laboratory
Los Alamos, NM, USA
gransom@lanl.gov

Jun Li
CUNY Queens College and Graduate Center
New York, NY, USA
jun.li@qc.cuny.edu

Haryadi S. Gunawi
University of Chicago
Chicago, IL, USA
haryadi@cs.uchicago.edu

ABSTRACT

Multi-level erasure coding (MLEC) has seen large deployments in the field, but there is no in-depth study of design considerations for MLEC at scale. In this paper, we provide comprehensive design considerations and analysis of MLEC at scale. We introduce the design space of MLEC in multiple dimensions, including various code parameter selections, chunk placement schemes, and various repair methods. We quantify their performance and durability, and show which MLEC schemes and repair methods can provide the best tolerance against independent/correlated failures and reduce repair network traffic by orders of magnitude. To achieve this, we use various evaluation strategies including simulation, splitting, dynamic programming, and mathematical modeling. We also compare the performance and durability of MLEC with other EC schemes such as SLEC and LRC and show that MLEC can provide high durability with higher encoding throughput and less repair network traffic over both SLEC and LRC.

CCS CONCEPTS

• **Computer systems organization → Reliability**; **Redundancy**; **n-tier architectures**; • **Computing methodologies → Simulation evaluation**.

KEYWORDS

Data Centers, HPC Storage, Scalable Storage, Reliability, Data Protection, Erasure Coding, System-Design Tradeoffs

ACM Reference Format:
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 *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC '23), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3581784.3607072

1 INTRODUCTION

Large-scale data centers store a huge amount of user data in a massive number of disks and require redundancy approaches such as erasure coding (EC) to protect them from disk failures [1–8]. The sheer size, scale, complexity, and layering of distributed mass-capacity storage keep growing and have never stopped. Figure 1 shows that the number of disks managed in Backblaze and US DOE laboratories keeps increasing and the per-disk capacity also keeps growing for both max available capacity and average capacity of sold disks. Such extreme scales lead to more frequent disk failures and longer time to rebuild a failed disk.

For managing tens or hundreds of thousands of disks, the classical single-level erasure coding (SLEC) no longer scales and cannot provide a good balance between minimizing repair overhead and maximizing rack/enclosure-level failure tolerance. Multi-level erasure coding (MLEC), which performs EC at *both* network and local levels, becomes a popular choice for several reasons. **(a)** MLEC is a hybrid of the network- and local-level SLEC schemes, hence gaining the benefits of the two worlds. Local SLEC only performs EC inside a rack/enclosure and thus cannot tolerate rack/enclosure failures [9–11]. Network SLEC tolerates rack failures but requires cross-rack network traffic for repairing lost chunks [12–14]. MLEC, on the other hand, can repair most disk failures locally without interfering user network traffic while at the same time being able to tolerate rack failures. **(b)** MLEC's performance scales better. With tens of thousands of disks, deploying more parity chunks and wider stripes to achieve higher durability will lead to higher encoding

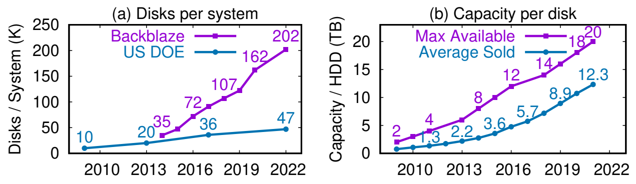

Figure 1: Storage scaling over the years.

computation overhead. However, the 2-level nature of MLEC can provide high durability with less encoding overhead than SLEC. **(c)** MLEC is stackable and easy to deploy/scale in practice. Since storage vendors sell RBODs (reliable bunch of disks) with EC controllers inside, larger-scale customers can build network-level EC on top of the local RBODs. **(d)** MLEC is more configurable. Customers can choose how many parities and what kind of chunk placement scheme to use at each level that fit their goals and constraints.

MLEC has seen large deployments in the field, including in HPC data centers in national laboratories [15], enterprise-grade storage softwares [16], and commercial storage systems [17]. However, based on literature study and personal communications, there is no in-depth study of design considerations for MLEC at scale. Many research questions remain unanswered. What are the possible chunk placement schemes for MLEC at scale? What are their pros/cons in terms of performance and durability? What are the types of failure modes an MLEC system can face? Can we introduce advanced repair methods that are optimized for every specific scheme and failure mode? What are the implementation requirements for advanced repairs? Though other works analyze hierarchical RAID for small-scale systems [18–22], we have not seen any work answering the questions above or studying design considerations of MLEC for large-scale systems.

In this paper, we provide, to the best of our knowledge, the most comprehensive design considerations and analysis of MLEC at scale that addresses the questions above. More specifically, we present the following contributions.

(1) We introduce the design space of MLEC in multiple dimensions, including various code parameter selections, chunk placement schemes, and various repair methods from a simple and practical one to a more optimized one that leverages the multi-level EC but requires cross-level transparency.
(2) We quantify their performance (encoding throughput, repair network traffic, repair time, etc.) and durability (under independent and correlated failures). We show which MLEC schemes and repair methods can provide the best tolerance against independent/correlated failures and reduce repair network traffic by orders of magnitude.
(3) To achieve all of the above, we use various evaluation strategies including simulation, splitting, dynamic programming, and mathematical modeling. We build, to the best of our knowledge, the first sophisticated MLEC simulator (in almost 13,000 lines of code (LOC)) that allows us to measure MLEC performance and durability at scale (over 50,000 disks), with many capabilities such as simulating disk failures (based on distributions, rules, or real traces), combining multi-level clustered/declustered placements, expressing failure tolerance, and executing complex repairs.

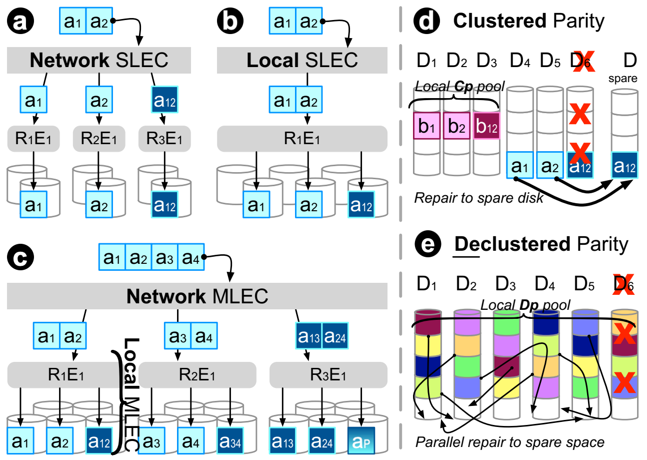

Figure 2: SLEC vs. MLEC logical view (§2.1). *The figures show (a) a network SLEC, (b) a local SLEC, and (c) an MLEC. Light-colored boxes (e.g., a_1, a_2) are data chunks and dark-colored boxes (e.g., a_{12}, a_{24}) are parity chunks. "R" and "E" respectively denote racks and enclosures. Figures (d) and (e) differentiates local clustered and declustered parity placements.*

(4) We also compare the performance and durability of MLEC with other EC schemes such as SLEC and LRC [23] and show that MLEC can provide high durability with higher encoding throughput and less repair network traffic over both SLEC and LRC.

Configuring or extending an MLEC design at extreme scales can be costly without proper understanding of the implication of the proposed changes. We hope our analysis (along with the simulation code [24] and evaluation artifact [25] that we have released) will enable engineers and operators of extreme-scale EC systems to have a comprehensive knowledge of the (dis)advantages of various MLEC schemes.

2 MLEC DESIGN

We begin by describing the MLEC design, starting from the logical and physical views (§2.1-2.2) to the failure modes and possible repair methods (§2.3-2.4). To ease readers in finding definitions and descriptions, we use bold text for findings, the first mentions of figure references, and important terms.

2.1 MLEC Basics and Logical View

We begin with showing the *logical view* of the MLEC architecture by comparing it with basic SLEC architectures in **Figure 2**. For simplicity, not all physical elements are shown yet. We use the **(k+p)** notation to describe an SLEC setup with *k* data and *p* parity chunks. For MLEC, we use the **$(k_n+p_n)/(k_l+p_l)$ notation** where *n* and *l* respectively stand for "network" and "local."

Network SLEC: Figure 2a shows a simple (2+1) *network* SLEC with three enclosures. When **user data** arrive (a_1 and a_2), the storage server builds the parity chunk (a_{12}) and sends each of these chunks to a separate rack, and in each rack the chunk might go to a different enclosure (*e.g.*, a_1 might go to rack R_1 enclosure E_1, a_2 to R_2E_3, and a_{12} to R_3E_2. But for simplicity, a_1, a_2, a_{12} all go to E_1 in the figure). An **enclosure** is a collection of disks stored within the same **rack**. A "disk" can be an HDD, SSD, or other types of drives. Network-level SLEC can tolerate rack/enclosure-level failures but requires cross-rack network traffic for every repair.

Local SLEC: Figure 2b shows a (2+1) *local* SLEC with one enclosure. The storage server picks an enclosure and simply forwards the entire user data stripe to the enclosure-level controller, which then builds the parity chunk and writes all the chunks to different disks in the enclosure. Local SLEC can tolerate disk failures but not rack/enclosure-level failures.

MLEC: Figure 2c shows a (2+1)/(2+1) MLEC architecture, which is a *combination of network and local* SLEC architectures. Note that the k_n and k_l do *not* have to be the same, but we use k_n=k_l=2 here for simplicity. Upon receiving a full data stripe (four chunks, from a_1 to a_4), the storage server splits it to two **network data chunks** (a_1a_2 and a_3a_4) and build one **network parity chunk** ($a_{12}a_{34}$). Note here, a network-level chunk contains two local-level chunks. The server then distributes them across different enclosures in separate racks. Each enclosure takes the data and splits it to **local data chunks** (*e.g.*, a_1a_2 is split to a_1 and a_2 chunks), computes the **local parity chunk** (*e.g.*, a_{12} in R_1E_1), and sends them to three different disks in the enclosure.

Overall, the local-level MLEC manages the **local stripes** (*e.g.*, a_1–a_2–a_{12} is a local stripe with three local chunks). Likewise, the network-level MLEC is responsible for managing the **network stripes** where each network stripe contains multiple local stripes (*e.g.*, $a_1a_2a_{12}$–$a_3a_4a_{34}$–$a_{13}a_{24}a_P$ is a network stripe containing three local stripes). a_P is the parity of a_{13} and a_{24}.

Clustered vs. declustered parity (Cp vs. Dp): Now let's look at the local placement, where one can deploy the conventional clustered or declustered parity placement. In the clustered parity ("**Cp**") placement, every (k+p) disks will form a **pool**. In **Figure 2d**, the 6 disks in one enclosure form *two* (2+1) **local-Cp** pools. Here, a local stripe must go to a specific pool, *i.e.*, a stripe either has no chunk in the pool, or has all the chunks residing in the pool. When a disk in a local-Cp pool fails, the local repairer reads from the k (two) surviving disks to reconstruct the lost data and write to a new spare disk. The rebuild time is *bottlenecked* both by the read bandwidth of only the participating disks (two disks here) and by the single disk's write bandwidth (to the one spare disk).

In order to improve the rebuild rate, declustered parity ("**Dp**") placement was proposed. For brevity, we will not discuss the detailed layouts, as they can be found in the literature [26–31]. However, the important part is that a **local-Dp** pool should have *(much) more than* (k+p) disks. In **Figure 2e**, *all* the 6 disks in the enclosure form only one local-Dp pool. Here, the data, parities, and spare space are pseudorandomly spread (declustered) across all the disks. When a disk fails, all the surviving disks in the large pool participate in both reading and writing which leads to faster repair rate. Later on, the admin can bring in a new disk and rebalances the data in the background.

Declustered parity placement can also be applied to network SLEC. In *network-Dp* SLEC, the entire system is treated as a pool, and each chunk in the stripe is placed pseudorandomly in a separate rack. When a disk fails, all the surviving disks in the system can participate in the repair, utilizing the network bandwidth of all the racks in the system to speed up the repair.

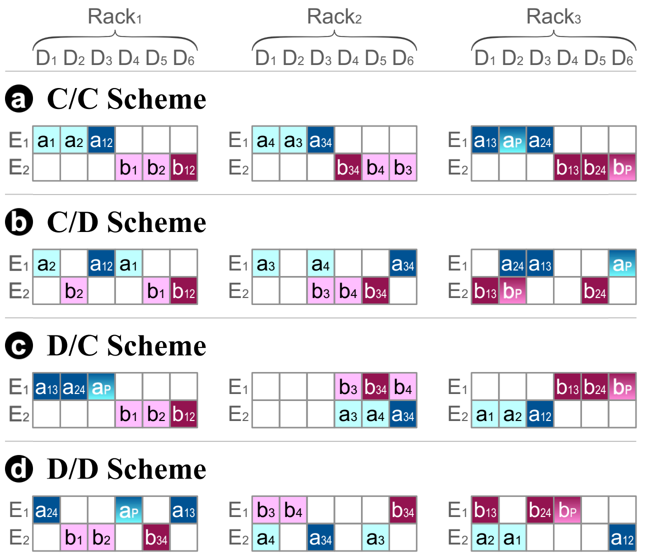

Figure 3: Four MLEC schemes and their physical views (§2.2). *The figure shows three racks, each with two enclosures, each with six disks. For simplicity, we only show chunk per disk (no disk cylinders); e.g., a_1 chunk is in Rack R_1, Enclosure E_1, Disk D_1.*

2.2 MLEC Schemes and Physical View

Given the two levels (network and local) and the two chunk/parity placements (clustered and declustered), we can permute them into four basic placement schemes (or "MLEC schemes" for short). Below we define each of them, using **Figure 3** to illustrate the physical views. Again, for simplicity, we use a (2+1)/(2+1) MLEC, *i.e.*, k_n=2, p_n=1, k_l=2, and k_l=1. Hence, we show 3 racks ($R_1$to R_3), where each rack contains 2 enclosures (E_1 and E_2) and each enclosure contains 6 disks (D_1 to D_6).

Clustered-clustered ($^C/_C$) scheme: In **Figure 3a**, this simplest scheme performs clustered parity at both network and local levels. A (2+1) local stripe is mapped to a local-Cp pool, containing adjacent k_l+p_l disks; for example, $a_1a_2a_{12}$ and $b_1b_2b_{12}$ are mapped to two different local pools, each with three consecutive disks. Moving up to the network level, every k_n+p_n enclosures at the same position across the three racks form a **network pool** for (2+1) network stripes. For example, for the network stripe $a_1 \ldots a_P$, they all have to reside in the same local-Cp pool position in the three E_1enclosures across the three racks.

Clustered-declustered ($^C/_D$) scheme: In **Figure 3b**, this scheme performs a network clustered and local declustered placements. Starting from the top, all the data and parity chunks of a network stripe still have to reside within the same local-Dp pool position within the same enclosure position in each rack. For example, $a_1 \ldots a_P$ are mapped to the first local-Dp pool in enclosure E_1 across the three racks. Locally, as explained before, the local-Dp pool has 6 disks. The chunks of a local stripe are pseudorandomly spread across the 6 disks (*e.g.*, a_2, a_1, and a_{12} chunks are mapped to disks D_1, D_3, and D_4, respectively), but the chunks in the stripe cannot go to the same disk (to tolerate disk-level failure).

Declustere-clustered ($^D/_C$) scheme: Reversing the previous scheme, now we have the network level performing declustered

Local level failures
• **A failed chunk**: A lost (but may be recoverable) chunk due to a disk failure.
• **An affected local stripe**: a local stripe with any number of chunk failures.
• **A locally-recoverable local stripe**: A local stripe containing 1 to p_l failed chunks.
• **A lost local stripe**: A local stripe containing p_l+1 or more failed chunks but may still be recoverable from the network level.
• **A catastrophic (locally-unrecoverable) local pool**: A local pool with 1 or more lost local stripes, *e.g.*, in (10+2)/(17+3) MLEC, a local pool with 4 disk failures is not recoverable locally and requires network repair.
Network level failures
• **An affected network stripe**: a network-wide stripe with any number of lost local stripes.
• **A recoverable network stripe**: A network stripe containing 1 to p_n lost local stripes.
• **A lost network stripe (a data loss)**: A network stripe with p_n+1 or more lost local stripes.

Table 1: MLEC failure modes (§2.3).

parity. That is, the local-stripes of a *network* stripe will be pseudo-randomly spread across the enclosures within the network pool, but they cannot go to the same rack (to tolerate rack-level failure). For simplicity, **Figure 3c** shows a network pool containing only six enclosures across the three racks. The network stripe $a_1 \ldots a_P$ is split to three local stripes stored in different enclosure positions in the three racks (*e.g.*, the local stripe $a_1a_2a_{12}$ is mapped to enclosure E_2 in rack R_3). At the local level, $^D/_C$ follows $^C/_C$, *i.e.*, a local stripe goes to a local-Cp pool.

Declustered-declusted ($^D/_D$) scheme: Finally, in this last scheme, we have declustered placements in both network and local levels. For example, in **Figure 3d**, just like in the previous figure/scheme, the local stripe $a_1a_2a_{12}$ is mapped to enclosure E_2 in rack R_3, but now the chunks of this local stripe are scattered across the 6 disks in the local-Dp pool.

In large-scale deployments, in network clustered ($^C/_*$)[1] schemes, every k_n+p_n inter-rack local pools in the same enclosure position will form a network pool, hence the total rack count must be a multiple of k_n+p_n. However, in network declustered ($^D/_*$) schemes, a network pool usually contains much more than but does not have to be a multiple of k_n+p_n racks. Likewise, in local clustered ($^*/_C$) schemes, a local pool contains k_l+p_l disks, and hence an enclosure must have a multiple of k_l+p_l disks. However, in local declustered ($^*/_D$) schemes, a local pool usually contains much more than but does not have to be a multiple of k_l+p_l disks.

2.3 Failure Modes

Given the more complex chunk placements, MLEC can face various failure modes, as listed in **Table 1**, which we will use heavily throughout the paper. With the listed definitions, we now can derive the data loss conditions, which vary across the MLEC schemes.

A **data loss** is defined as the loss of a network stripe, more specifically the loss of p_n+1 local stripes. In network-Cp ($^C/_*$) schemes, only p_n+1 catastrophic local pools in the same network pool can cause a network stripe to have p_n+1 lost local stripes. Since a $^C/_*$ system can have many network-level pools, p_n+1 catastrophic local pools that are scattered in multiple network-level pools will not cause data loss. In network-Dp ($^D/_*$) schemes, since there is only one network pool in the system, any arbitrary p_n+1 catastrophic local pools may lead to a lost network stripe with p_n+1 lost local stripes. However, the probability for such a lost network stripe to happen can be extremely low, depending on the actual chunk placement (which is pseudorandom) in network-Dp ($^D/_*$) schemes.

2.4 Repair Methods

When it comes to repair, the network pool level is more important. Repairing locally-recoverable pools is straightforward (similar to SLEC repairs in Figures 2d-e). The challenge is to recover[2] a **catastrophic (locally-unrecoverable) local pool** (defined in Table 1). For this, we introduce four possible local-pool repair methods applicable to all the MLEC schemes, as illustrated in **Figure 4**, from the simple to optimum ones, along with their pros and cons.

Repair All (R_{ALL}): In **Figure 4a**, the failures of disks D_1 and D_3 in rack R_1 caused a catastrophic local pool failure, thus chunks a_1 and a_2 need to be reconstructed. "Repair All" (R_{ALL}) is a method that simply *rebuilds the entire local pool* (*e.g.*, disks D_1 to D_6 in rack R_1) from the other healthy local pools in other racks (R_2 and R_3) via a network-level parity calculation. As the downside, it unnecessarily leads to a much higher amount of network traffic. However, R_{ALL} is common in deployment (*e.g.*, in MarFS [15]) because it is considerably the *easiest* to implement. That is, the network repairer does not need to know the layouts of the local part of the MLEC. The network-level sysadmins can use black-box/off-the-shelf RBODs (*e.g.*, CORVAULT [32], ZFS pools [33], and PowerEdge RAID [34]).

Repair Failed Chunks Only (R_{FCO}): Unlike R_{ALL}, R_{FCO} only rebuilds the failed chunks. For example, in **Figure 4b**, a_1 in rack R_1 is rebuilt by network parity calculation on a_3 and a_{13} from the other two racks (and similarly for a_2). By doing this, R_{FCO} reduces the network repair traffic. Although it seems simple, R_{FCO} is less straightforward to implement. It requires the local/enclosure-level repairer to report which chunks have failed and coordinates with the network-level repairer. This is one reason why repair methods like R_{FCO} are not supported by many existing RAID systems. For example, ZFS [33] handles its own block device mappings internally, and does not readily expose the mapping information. Therefore, when a ZFS local pool fails, it is often impossible for the network-level repairer to know which chunks have failed. Thus, although R_{FCO} is more efficient than R_{ALL}, R_{FCO} requires a proper API design and metadata management across the local and network levels.

Repair Hybrid (R_{HYB}): R_{HYB} repairs the failed chunks in a hybrid way using both network and local repairs. To show this optimization, **Figure 4c** shows a slightly different layout, where two failed chunks a_2 and b_2 are stored in the same failed disk D_3 in rack R_1(plus another failed chunk a_1 in disk D_1). Here, R_{HYB} basically analyzes every failed chunk and asks whether a network repair is needed or not. The loss of a_1 and a_2 form a *lost local stripe*,

[1]$*$ **is a don't-care symbol.**

[2]We use recover/repair/rebuild/reconstruct interchangeably.

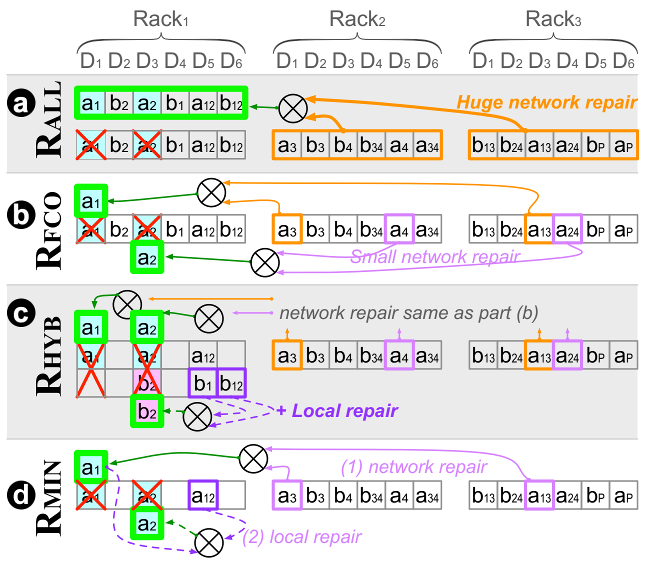

Figure 4: Four repair methods, R_{ALL} to R_{MIN} (§2.4). *For simplicity, we only show a (2+1)/(2+1) $^C/_D$ MLEC scheme. Also, not all the figures use the same chunk locations.*

hence they *cannot* be repaired locally and require the network repair. However, for b_2, it can be rebuilt locally because b_1 and b_{12} are still available in the surviving disks (*i.e.*, a *locally-recoverable stripe*). As we measure and explain later in Section 4.2, R_{HYB} works well for local-Dp ($^*/_D$) schemes.

Repair Minimum (R_{MIN}): Taking the insight from the previous method, this last method incurs the minimum amount of network repair traffic. It does so in two stages. First, it finds all the lost local stripes and reads the minimum number of chunks over the network such that the stripes transition to locally-recoverable local stripe. Second, all the locally-recoverable local stripes can be rebuilt locally. For example in **Figure 4d**, the lost local stripe $a_1a_2a_{12}$ is partially repaired by rebuilding a_1 first (by xor-ing a_3 and a_{13}) over the network. Now, a_2 is still lost but can be rebuilt locally by xor-ing a_{12} and the recently-rebuilt a_1.

3 METHODOLOGY

To perform in-depth analysis for various MLEC schemes and repair methods, we use the four following evaluation strategies.

Simulation: To quantify performance and reliability, one can start with a mathematical model. However, our work introduces complex repair methods that are hard to model. For this reason, we build a sophisticated MLEC simulator (in almost 13 kLOC) with many capabilities such as simulating disk failures (based on distributions, rules, or real traces), combining multi-level (de)clustered placements, expressing failure tolerance, and executing complex repairs. We use this simulator to measure repair traffic, repair time, and system durability in Sections 4.1.2, 4.1.3, 4.2.1, 4.2.2, 5.1.4, 5.2.4.

Splitting (multi-stage simulation): A uniqueness of our work is the focus on extreme-scale deployments (*e.g.*, >10k disks), which require protection from a large number of parities. However, to estimate high durability, a simulation must run a large number of iterations to capture even one system data loss event (it will take years even with a 200-core simulation). To reach rare events faster, we adopt the splitting method [35, 36]. First, we simulate the durability of a single local pool using regular simulation and collect local pool failure samples. Second, we systematically inject catastrophic local pool failures from the samples at MLEC level. We use this to evaluate high durability in Sections 4.2.3, 5.1.2, and 5.2.2.

Dynamic programming: While splitting is efficient to measure high durability under independent failures, it is hard to do the same under correlated failure bursts. This is because each local pool's durability is not deterministic but correlated to other local pools' durability. Thus, we use dynamic programming, specifically by using it to count the number of all the possible disk failure layouts under a certain correlated failure burst scenario, and then count how many such failure layouts could cause a data loss in MLEC. We use this strategy for measuring the probability of data loss (PDL) under correlated failure bursts in Sections 4.1.1, 5.1.3 and 5.2.3.

Mathematical model: For thoroughness, we also build a mathematical model to verify our simulation strategies, but we can only do so for the simplest repair method (R_{ALL}). We choose to use Markov Chain model as it's commonly used to analyze durability of SLEC systems [37–39]. To model the MLEC system, we first took existing SLEC durability models that use Markov chain models and probability theory [37–40], and then we iteratively apply the model to network-level MLEC by treating a local pool like a disk.

(Setup): We use the following setups, which mimic real large-scale deployments [15, 41, 42]. *Datacenter setup:* We simulate 57,600 disks across 60 racks, with 8 enclosures per rack, 120 disks per enclosure, 20 TB per disk, and a chunk size of 128 KB. *MLEC configuration:* We use a **(10+2)/(17+3)** MLEC. For $^*/_C$ schemes, a local-CP pool contains exactly 20 disks. For $^*/_D$ schemes, given the local declustered approach, a local-DP pool contains 120 disks. *Available repair bandwidth:* The *raw* bandwidth is set to be 200 MB/s for per-disk I/Os and 10 Gbps per rack for cross-rack network, respectively. However, for repairs, disk and network traffics are both capped at 20% of their respective raw bandwidth. Thus, we use the term "available repair bandwidth" to reflect the resulting data repair bandwidth that is subject to this policy. *Fault simulation:* To simulate a catastrophic local pool failure, we generate p_l+1 disk failures simultaneously. However, to measure long-term durability (*e.g.*, over one year), we generate random disk failures independently following an exponential distribution with an annual failure rate (AFR) of 1%. *Failure detection time:* We also follow previous works [23, 43] that use 30 minutes to detect each failure and trigger the repair.

4 MLEC ANALYSIS

We now present an in-depth analysis of performance and durability implications of various MLEC schemes in Section 4.1 and repair methods in Section 4.2.

4.1 Analysis of MLEC Schemes

First, we analyze the four MLEC schemes ($^C/_C$ to $^D/_D$), particularly the impact of their chunk/parity placements on the probability of data loss under correlated failures (§4.1.1), repair speed (§4.1.2), and probability of catastrophic local failure (§4.1.3).

4.1.1 *PDL under Correlated Failures.*

We begin with analyzing the impact of different MLEC schemes on the probability of

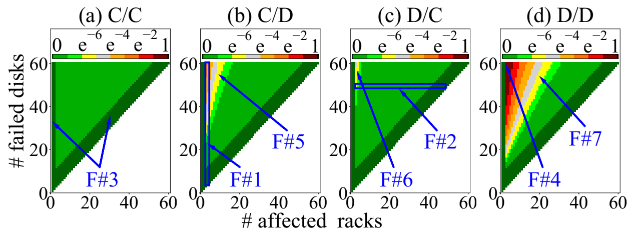

Figure 5: PDL under correlated failures (§4.1.1). *The square color represents the PDL of the MLEC scheme when a total of y simultaneous disk failures are randomly scattered across x racks.*

data loss (PDL) under *a wide range* of failure burst topologies, from scattered disk failures across many racks to highly correlated failures localized in a single rack. **PDL** is defined as the probability that a storage system unrecoverably loses any data, and **failure bursts** refer to failures that are temporally correlated and happen concurrently at the same time or within a small time window [37].

Figure 5 shows heatmaps of the PDL, where greenish squares (near the value of 0) represent high durability and reddish squares (near the value of 1) represent low durability. We vary the number of disk failures (in the y-axis) and spread them across one or more racks (in the x-axis). For example, $\{y = 60, x = 60\}$ implies that every rack has a single disk failure (*i.e.*, scattered failures), while $\{y = 60, x = 1\}$ implies that only one rack experiences 60 disk failures. The four subfigures show the impact of these various failures on C/C, C/D, D/C, and D/D schemes, respectively.

Below we present our findings "**F#1-7**" in Figure 5. Findings #1-4 are applicable to all four schemes, but for presentation clarity, we spread them across the four subfigures. Findings #5-7, however, are unique to the labeled schemes, respectively. We believe Findings #2-#7 were never reported before.

Finding #1: *When a failure burst happens in at least p_n+1 racks, the more failed disks in those racks, the higher the PDL.* For example, in the highlighted vertical area in Figure 5b, p_n is 2, but we have 3 racks ($x = 3$) experiencing failures. If the number of failed disks goes up (in the y-axis), it will cause more local stripe failures. As a result, the PDL will go up (greenish to reddish squares), and data loss will happen if p_n+1 or more local stripe failures happen within a network stripe (as the network stripe cannot recover).

Finding #2: *MLEC is more robust to scattered failures.* As highlighted in the horizontal area in Figure 5c, when a fixed number of disk failures happen concurrently, the more racks they are scattered to, the lower the PDL is. This is because each rack is more likely to have fewer disk failures, which is more tolerable by the local-level EC of MLEC.

Finding #3: *Full local failures can be recovered by the network-level EC up to some limit.* In Figure 5a, zero data loss can be guaranteed (PDL = 0) when the number of affected racks is smaller than or equal to 2. This is because a network stripe can survive any p_n rack failures, and p_n is 2 here. The PDL is also 0 when no more than x+8 disk failures are scattered in x racks. This is because a local (17+3) stripe can tolerate any 3 failures. Thus, x+8 disk failures in x racks can cause at most 2 lost local stripes in the same network stripe, which can be tolerated by the network-level (10+2) EC.

Finding #4: *MLEC is susceptible to data loss under highly localized failure bursts.* Deriving from previous findings, MLEC suffers the most when failures bursts happen in exactly p_n+1 racks, depicted in Figure 5d. In all the figures, we can also see that PDL is the highest when 60 disk failures happen concurrently in 3 racks.

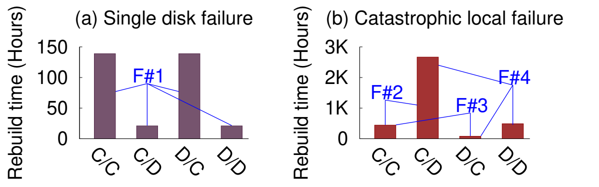

Figure 6: Repair time (§4.1.2). *Under (a) a single disk failure and (b) a catastrophic local failure.*

	Single disk failure		Local pool failure	
MLEC Schemes	Disk size (TB)	Avail. Repair BW (MB/s)	Pool size (TB)	Avail. Repair BW (MB/s)
C/C	20	40	400	250
C/D	20	264	2400	250
D/C	20	40	400	1363
D/D	20	264	2400	1363

Table 2: Repair size and available repair bandwidth (§4.1.2). *Under (a) single disk failure and (b) catastrophic local failure.*

Finding #5: *C/D has lower tolerance than C/C under more localized failure bursts.* By comparing the area pointed to by Finding #5 in Figure 5b and the same area in Figure 5a, we can see that C/C has better tolerance than C/D when failure bursts happen in more than p_n racks. This is because a local-Dp pool can only tolerate p_l arbitrary concurrent disk failures out of many (let's say D_l) disks. On the other hand, a local-Cp pool can tolerate up to p_l disk failures out of (k_l+p_l) disks, where usually D_l>k_l+p_l. Therefore, when the same number of random disks in a rack fail concurrently, it's more possible to cause local pool failures in C/D than in C/C, and more frequent local pool failures in turn make system-wide data loss more likely.

Finding #6: *Likewise, D/C has lower tolerance than C/C.* Similar to above, when p_n+1 or more racks have failures, D/C performs worse than C/C, yet for a different reason. Whenever more than p_n racks have local pool failures, no matter which local pool in the rack fails, D/C can lose data. On the other hand, C/C experiences data loss only when more than p_n local pools in the same network-level pool fail.

Finding #7: *D/D has the worst data durability under failure bursts.* In Figure 5, D/D has the most reddish squares (higher PDL) among all the schemes. This is because the local-Dp scheme makes local pool failures more likely, and local pool failures in the network-Dp scheme are more likely to cause data loss. Even when failures are roughly scattered, unlike other schemes, D/D has a lower chance of survival, because its local-Dp pools are more possible to fail compared to local-Cp pools in C/C and D/C. Furthermore, D/D can lose data when any arbitrary p_n+1 local pools in separate racks fail, while C/C and C/D only lose data when p_n+1 local pools in the same network-level pool fail, which is less likely to happen.

In conclusion, different MLEC chunk placement schemes provide different tolerance against correlated failure bursts. Among them, C/C performs the best while D/D has the largest probability of data loss. However, C/C's repair rate unfortunately is not as fast as the other schemes, as we will dissect in the next section.

4.1.2 ***Repair Speed.*** We now analyze how long data repair will take under various MLEC schemes. Specifically, we analyze the time of repairing (a) a single disk and (b) a catastrophic local failure. *This section starts with the simplest method, Repair-All (R_{ALL})*, but later Section 4.2 uses all the repair methods.

Figure 6 shows the rebuild time of the four MLEC schemes under both failure conditions. To explain the rebuild time in the figure, **Table 2** provides further information on the amount of data to rebuild and the available repair bandwidth, which also depends on how many disks/local pools/racks participate in the repair as explained in earlier in the "R_{ALL}" paragraph of Section 2.4. We now elaborate findings "**F#1-4**" in Figure 6.

Finding #1: *In repairing a single disk failure, local declustered placement in $^C/_D$ and $^D/_D$ makes rebuilding fast.* Figure 6a shows that $^C/_D$ and $^D/_D$ are 6x faster compared to $^C/_C$ and $^D/_C$. This is because, when a disk in a 120-disk local-Dp pool fails, the local repairer can in parallel read the healthy chunks from and write the reconstructed chunks to spare spaces on *all the* 119 surviving local disks. On the other hand, the local-Cp repairer reads from 19 surviving disks and rebuilds to *only* 1 spare disk. Since it only has at most 20 disks to participate in the local repair, its available repair bandwidth is lower. In Table 2, the single disk repair bandwidth in $^C/_C$ and $^D/_C$ is around 6x lower than that of $^C/_D$ and $^D/_D$.

Finding #2: *In repairing a catastrophic local failure, $^C/_D$ takes the longest time due to its larger local pool size.* While in Figure 6a, $^C/_D$ is faster than $^C/_C$ in rebuilding a single disk failure, it is the contrary under a catastrophic local failure, as shown in Figure 6b. This is because $^C/_D$ has a local-Dp pool whose size is much larger than that of a local-Cp pool, leading to more data to reconstruct across the network with limited network bandwidth. As detailed in Table 2, the local-Dp ($^*/_D$) pool size is 2400 TB while the local-Cp ($^*/_C$) pool size is only 400 TB.

Finding #3: *In repairing a catastrophic local failure, $^D/_C$ is the fastest scheme.* The speed-up comes from the network-level declustered chunk placement. When reconstructing the target local pool, the repairer needs to read chunks from other local pools. Thanks to the network-level declustering, the chunks are spread across all the other 59 racks and the rebuilt data can be written to spare spaces in all the 60 racks, including the rack that contains the failed local pool. This in turn gives a 5x repair rate compared to $^C/_C$, which has only 12 (from 10+2) racks participating in the repair at most. As shown in Table 2, $^C/_C$'s local pool repair bandwidth is 250 MB/s while $^D/_C$'s can be as high as 1,363 MB/s.

Finding #4: *$^D/_D$ is faster than $^C/_D$, but slower than $^D/_C$ and slightly slower than $^C/_C$.* Still in Figure 6b, compared to $^C/_D$ (the slowest of all), $^D/_D$ is around 5x faster due to the network declustering. However, with its local pool that is 6x larger in size compared to that of $^D/_C$, $^D/_D$ is around 6x slower than $^D/_C$. As a result, $^D/_D$ takes a bit longer than $^C/_C$ as the repair overhead caused by the large local-Dp pool dominates here. However, in a cluster with more racks and smaller local-Dp pool size, or in a cluster with "unlimited" network bandwidth, $^D/_D$ could be faster than $^C/_C$ in repairing a catastrophic local pool. This is because $^D/_D$ theoretically can read from and write to all local pools.

In conclusion, under a single disk failure, $^C/_D$ and $^D/_D$ have the fastest repair rate, but $^D/_C$ is the fastest under a catastrophic local failure. However, we emphasize the tradeoffs again that all these speed-ups are obtained at the cost of a higher probability of data loss against correlated failure bursts (as explained in Section 4.1.1 where $^C/_C$ has the lowest PDL). We also have shown other tradeoffs where single disk repair is faster (*e.g.*, $^C/_D$ is faster than $^C/_C$ in Figure 6a) but at the cost of slower local pool repair (*e.g.*, $^C/_D$ is slower than $^C/_C$ in Figure 6b).

4.1.3 ***Local Failure Probability.*** Prior sections have established that a catastrophic local pool failure is an *Achilles' heel* of MLEC because it will lead to a huge amount of network traffic with limited available bandwidth. Even worse, when p_n+1 catastrophic local failures happen concurrently, the system will lose data. Thus, we now ask what the probability of catastrophic local failure is in different MLEC schemes.

Figure 7 shows that *fortunately the probability of the catastrophic local failure is low in all MLEC schemes.* For example, the probability with $^C/_C$ and $^D/_C$ is lower than 0.001% per year. Even better, the probability is almost 0.00001% with $^C/_D$ and $^D/_D$. There are two main reasons why the latter is better by orders of magnitude. First, a local-Dp pool has higher durability than a local-Cp pool. This is not only because the local-Dp pool has a higher disk repair rate, but also because it has less high-priority stripes (i.e. stripes that have multiple failed chunks), which can be prioritized and repaired quickly. Accordingly, a local-Dp pool is more durable in our setup. Second, since the local-Dp pool size (120) is larger than the local-Cp pool size (20), the system has fewer local-Dp pools. Thus, the probability of an arbitrary local-Dp pool failing is lower.

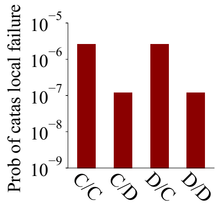

Figure 7: Prob. of catastrophic local failure.

4.2 Analysis of Repair Methods

Previous section brings the good news that the probability of the catastrophic local failure is low in general. However, as alluded to in Section 4.1.2, the repair will take a large amount of time under a straightforward repair method, R_{ALL}. Thus, in the following sections, we extend the evaluation to include R_{FCO}, R_{HYB} and R_{MIN}'s performance in repairing a catastrophic local pool failure.

4.2.1 ***Cross-Rack Repair Traffic.*** We begin with quantifying the cross-rack network traffic of the four repair methods (R_{ALL} to R_{MIN}) in all the four MLEC schemes ($^C/_C$ to $^D/_D$), as shown in **Figure 4.2.1** with the findings **F#1-4** described below.

Finding #1: *R_{ALL} is the simplest to implement but results in the most network traffic.* As elaborated in Section 2.4, R_{ALL} does not require the network-level and local-level repairers to be aware of each other, and hence simple to implement. As an implication, however, R_{ALL} needs to repair the entire local pool, as opposed to just the failed data in the local pool, which in turn leads to unnecessary work. R_{ALL} also results in much more network traffic in local-Dp ($^*/_D$) than in local-Cp ($^*/_C$) schemes (26,400 vs. 4400 TBs in the figure). This is because local-Dp has a larger local pool to reconstruct (120 disks) than local-Cp (20 disks).

Finding #2: *R_{FCO} significantly improves upon R_{ALL}.* This is because, instead of repairing the entire pool, R_{FCO} only repairs the failed chunks. The reduction is more apparent for local-Dp ($^*/_D$)

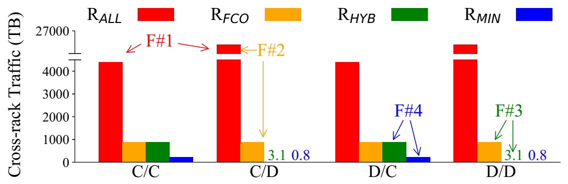

Figure 8: Cross-rack network traffic (§4.2.1). *The figure shows the cross-rack traffic (in TB) generated by the four different repair methods (R_{ALL} to R_{MIN}) on four MLEC schemes.*

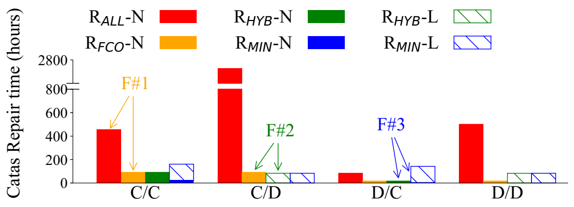

Figure 9: Repair time (§4.2.2). *The figure shows the network-level (-N) and local (-L) repair time with solid and striped bars, respectively. When the solid bars are not visible, the numbers are very small.*

schemes due to larger local pool sizes (*e.g.*, from 26,400 to 880 TB in $^{C}/_{D}$). As alluded to in Section 2.4, an MLEC deployment with two different vendors might need to implement some APIs to pass failure information across the two layers.

Finding #3: *R_{HYB} (a hybrid local and network repair) further reduces cross-rack repair traffic for local-Dp ($^{*}/_{D}$) schemes.* For $^{C}/_{D}$ and $^{D}/_{D}$, R_{HYB} only transfers 3.1 TB, much fewer than 880 TB in R_{FCO}. This is because in a local-Dp pool, when p_l+1 disks fail, only a small fraction of affected local stripes are lost local stripes which require network-level repair, and other affected local stripes that have less than p_l+1 chunk failures can be repaired locally. For local-Cp ($^{*}/_{C}$) schemes, the figure shows that R_{HYB} does not give advantage over R_{FCO}, because here we inject all the p_l+1 disk failures at the same time. Later in Section 4.2.3, we simulate different timings of failures.

Finding #4: *R_{MIN} provides the minimum cross-rack traffic among the four repair methods.* For all MLEC schemes, R_{MIN} reduces network traffic by 4x or more compared to R_{HYB}, thanks to the opportunistic 2-stage method in R_{MIN}. In this case, for example, instead of repairing all 4 failed chunks from the network level, R_{MIN} only repairs one failed chunk from each 4-chunk-failure stripe (the first stage) and after that rebuilds the three remaining failed chunks locally (the second stage).

4.2.2 *Repair Time.*

As the last section focuses on the network traffic size, we now measure the repair time and describe findings **F#1-3** in **Figure 9**. The first two repair methods, R_{ALL} and R_{FCO}, only employ network-level repair, while the last two, R_{HYB} and R_{MIN}, leverage local repairs, depending on the MLEC schemes, as we explain below.

Finding #1: *R_{ALL} imposes the longest time and R_{FCO} reduces the network-level repair time by 5-30x.* R_{ALL} reconstructs the entire local pool, leading to a slow repair (with network throttling) in all the MLEC schemes. However, R_{FCO} only reconstructs the 4 failed disks, resulting in much faster repair time.

Finding #2: *R_{HYB} reduces network repair time, but induces local repair time.* This is true for $^{C}/_{D}$ and $^{D}/_{D}$, wherein after the affected stripes are partially reconstructed via network repair, the local pool exits the catastrophic state and can be repaired locally. The local repair however can only read from the disks in the local pool (*i.e.*, less parallelism compared to reading from network-wide disks in other racks). For example, on $^{C}/_{D}$, R_{HYB} takes a similar amount of time as R_{FCO} to fully recover the data.

Finding #3: *While R_{MIN} transfers the minimum amount of data over the network, it can take even longer to repair the local pool.* In all the MLEC schemes, as R_{MIN} quickly repairs a less amount of data from the network level, it makes the failed local pool exit the catastrophic state faster, but could take longer time to fully repair all the failed disks. We would like to note that although the total repair time is longer due to local repair, R_{MIN} reduces network contention for foreground I/Os and improves the durability of the system (since the network-level EC is able to tolerate more catastrophic local failures after the fast network-level repair).

4.2.3 *Data Durability.*

Continuing what we built in the last section, we now analyze how different repair time from different methods would affect long-term data durability, as shown in **Figure 10** along with findings **F#1-4**. **Data durability** in one year is measured in *the number of nines* which is defined as $-\log_{10}$ (PDL), *e.g.*, 99.999% durability means 5 nines. In addition to the basic setups (§3), we also prioritize repairing local stripes with more failed chunks in local-Dp ($^{*}/_{D}$) schemes, and network stripes with more affected local stripes in network-Dp ($^{D}/_{*}$) schemes.

Finding #1: *Compared to R_{ALL}, R_{FCO} increases the durability by 0.9-6.6 nines.* R_{ALL}'s slow repair impacts durability, but R_{FCO}'s faster repair increases the durability. The increase is as high as 6.6 nines in $^{D}/_{D}$ due to two reasons. First, the repair time reduction is large in $^{D}/_{D}$ (Section 4.2.2). Second, when $^{D}/_{D}$ has p_n+1 catastrophic local pools, it might not have any lost network stripe because each catastrophic local-Dp pool only has a small number of lost local stripes, and the probability of a lost network stripe with p_n+1 lost local stripes is as low as 0.03% due to the network-Dp placement. However, R_{ALL} is not able to detect this information as it treats the entire local-Dp pool as lost. But again in R_{FCO}, the network repairer has the knowledge of which exact chunks are lost, and hence it can tolerate the cases where there are p_n+1 catastrophic local pools but no lost network stripes.

Finding #2: *R_{HYB} further increases the durability by 0.6-4.1 nines.* This is because R_{HYB}'s faster repair only rebuilds the lost local stripes. The increase is more apparent in $^{C}/_{D}$ and $^{D}/_{D}$ because each local-Dp pool only has a small portion of lost local stripes (due to the declustered placement). Note that R_{HYB} also increases the durability of local-Cp ($^{*}/_{C}$) schemes although R_{HYB} did not show repair traffic/time reduction in earlier sections. This is because previously we injected all the p_l+1 disk failures at the same time to reflect a catastrophic local pool failure. However, to measure long-term durability (Section 3), we let disks fail independently following exponential distribution, and thus disks usually fail at different times. Therefore, some disks could have been partially repaired when a catastrophic local pool failure happens. In such cases, only part of the affected local stripes are lost local stripes, and R_{HYB} can still deliver the traffic reduction for $^{*}/_{C}$ schemes.

Finding #3: *R_{MIN} further increases the durability by 0.1-1.2 nines.* R_{MIN} further increases the durability since it further decreases the

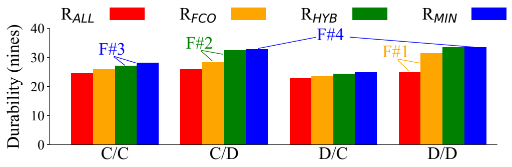

Figure 10: Durability (§4.2.3). *The figure shows the durability of MLEC schemes under various repair methods.*

repair time, especially for C/C. However, the increase is small in C/D and D/D. This is because their network repair is already fast (because the local declustered placement results in much fewer lost local stripes that require network repair), and thus the repair is bottlenecked by the time to detect the failure and trigger the repair.

Finding #4: *After all the optimizations, C/D and D/D provide the best durability while D/C provides the worst.* Among all MLEC schemes, C/D and D/D provide the best durability, because a local-Dp pool has higher durability than a local-Cp pool, thanks to the priority reconstruction in local-Dp. Moreover, a catastrophic local-Dp pool has fewer lost local stripes to repair compared to a catastrophic local-Cp pool, resulting in less network repair time, especially under R_{MIN}. Note that although D/D has a higher chance to have p_n+1 catastrophic local pools in a network pool compared to C/D, its disadvantage in terms of the durability is compensated as there are cases when D/D has p_n+1 catastrophic local pools but no lost network stripes, as mentioned in Finding #1 above. On the other hand, D/C provides the worst durability as it can lose data when any p_n+1 arbitrary local-Cp pools in separate racks fail catastrophically, and its benefit of fast repair from network-Dp is bottlenecked by the failure detection time.

5 VS. OTHER EC SCHEMES

We now evaluate MLEC with SLEC and LRC [23], in terms of their encoding throughput, durability, failure burst tolerance, and repair network traffic.

5.1 vs. SLEC

5.1.1 *Encoding Throughput.* **Figure 11** shows the single-core encoding throughput (the heatmap color) under various k and p configurations, in the x- and y-axis, respectively. We perform the measurement using the Intel ISA-L tool [44] on a single core of Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz.

Generally, *EC with larger values of k and p has lower encoding throughput.* When more parities (higher p) need to be computed, more computation overheads are introduced. On the other hand, with wider stripes (larger k), the encoding process might not be able to fit the required input data into CPU cache, which can degrade the encoding throughput [43]. This is a reason why SLEC is hard to scale. With MLEC, we can limit the number of parities and stripe width and attain better durability as we show next.

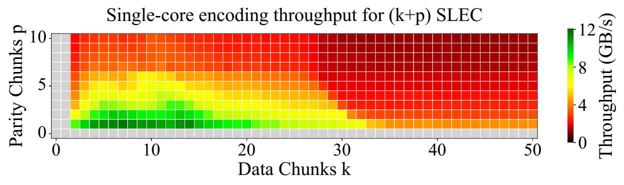

Figure 11: Encoding throughput for various (k+p) (§5.1.1).

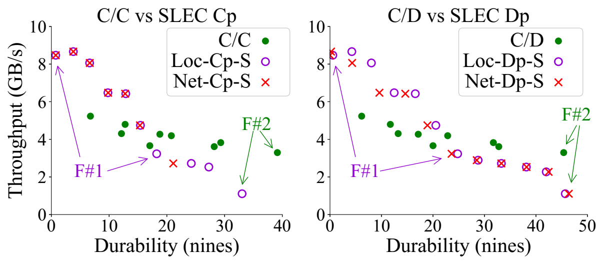

Figure 12: MLEC vs. SLEC durability/throughput tradeoff (§5.1.2). *A dot represents a specific configuration. For example, the green MLEC C/C and C/D dots come from various configurations such as (5+1)/(5+1), (10+2)/(17+3), and many others. For fairness, all the dots have a configuration with around 30% parity space overhead.*

5.1.2 *Durability vs. Throughput.* EC designers face performance-durability tradeoffs, for example more parities give higher durability, but lower encoding throughput. **Figure 12** quantifies this tradeoff with two main findings, **F#1-2**. Due to space constraints, we only show two MLEC schemes, (a) C/C and (b) C/D, vs. a local (de)clustered SLEC ("Loc-Cp-S" or "Loc-Dp-S", respectively) and a network (de)clustered SLEC ("Net-Cp-S" or "Net-Dp-S"), whose layouts were already presented in Section 2.1). For MLEC's repair method, we use R_{MIN}, the most optimized one. For fairness, all the points in the figure come from MLEC/SLEC configurations with a capacity (parity space) overhead of roughly 30%.

Finding #1: *For both MLEC and SLEC, higher durability leads to lower encoding throughput.* To achieve higher durability, both MLEC and SLEC need more parities, and to maintain the same capacity overhead, they will need a wider stripe.

Finding #2: *MLEC can provide high durability while maintaining higher encoding throughput.* At low durability (*e.g.*, <20 nines), MLEC's throughput is lower than SLEC, *however* at high durability (*e.g.*, >20 nines), MLEC can maintain almost the same throughput while at the same time increasing its durability dramatically, thanks to the two-level protection (p_l and p_n) and our repair optimizations. For example, at the two points pointed by F#2 in Figure 12a, a (17+3)/(17+3) C/C can reach 39-nine durability with 3GB/s throughput while a local (28+12) SLEC reaches 33-nine durability with only 1GB/s throughput. Increasing throughput can be done with more CPU cores, but would lead to higher hardware cost, and potentially extra overhead caused by imperfect parallelism.

5.1.3 *Failure Burst Tolerance.* Similar to the MLEC's failure burst analysis earlier in Figure 5 in Section 4.1.1, **Figure 13** shows the PDL of an (7+3) SLEC under correlated failure bursts with four possible chunk placements (local-Cp, local-Dp, network-Cp, and network-Dp). We describe their pros/cons. Again, by combining the two levels, MLEC hides each of the SLEC's limitations and gains the benefits of the two worlds.

Local SLEC is more susceptible to localized failure bursts. In **Figure 13a**, local-Cp SLEC can survive highly scattered failures when no more than y=x+p failures are scattered in x racks. However, it is susceptible to localized failure bursts since any p+1 disk failures in the same local-Cp pool can cause data loss. In **Figure 13b**, local-Dp

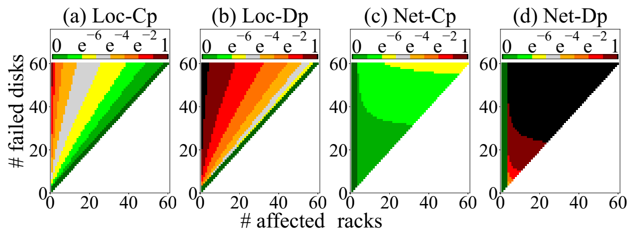

Figure 13: PDL of SLEC under correlated failures (§5.1.3). *Whose patterns (but not the actual values) can be compared with Figure 5. A total of y simultaneous disk failures are randomly scattered across x racks. The square color represents the PDL.*

SLEC performs even worse under localized failure bursts (high y, low x), since it has a larger local pool and thus has a higher chance to have p+1 disk failures in the pool, which can lead to data loss.

Network SLEC is more susceptible to scattered failure bursts. In **Figure 13c**, network-Cp SLEC performs well under localized failure bursts, and the PDL is 0 when no more than p racks have failures. However, it can lose data under scattered failures (high y, high x), which are more likely to have p+1 or more disk failures in the same network-Cp pool. In **Figure 13d**, network-Dp SLEC performs even worse under scattered failures, since it can lose data when any arbitrary p+1 disks in separate racks fail.

5.1.4 *Repair Network Traffic.*

Lastly, we analyze the repair network traffic, but due to space constraints, we do not show any figure. We find that network SLEC requires a huge amount of cross-rack repair network traffic, which can increase with more parities and wider stripes for higher durability. A (7+3) network SLEC requires hundreds of TB repair network traffic every day, which can largely interfere user network traffic. On the other hand, MLEC only requires a few TB repair network traffic every thousand of years, thanks to both the local protection and our repair optimizations.

5.2 vs. LRC

We now perform the same evaluation but with LRC, a popular EC approach that has been extensively studied in recent years [23, 43, 45–48]. We start with describing the layout differences.

5.2.1 *MLEC vs. LRC Layouts.*

A (k,l,r) LRC, in the first stage, divides k data chunks into l local groups and computes one local parity in each group, and in the second stage, computes r global parities from all the k data chunks [23]. **Figure 14** shows a (4,2,2) LRC. We treat LRC as a *one-level placement* EC, but with *two-stage* encoding. Unlike LRC, MLEC might fit better deployments where the two levels are managed by different organizations; for example, a large institution buys RBODs (local EC pools) from storage vendors and on top of them manages the network-level EC. LRC on the other hand might be more desirable to deployments that have direct access to and can manage all the disks.

Although MLEC and LRC both perform two stages of coding, they differ in several ways. **(a)** For the top-level encoding, each of MLEC's network parity is computed from only part of data chunks in the network stripe (*e.g.*, back in Figure 2.1c, a_{13} is computed from a_1 and a_3), but each of LRC's global parity is computed from all the data chunks in the stripe (*e.g.*, in Figure 14, a_P and a_Q are computed based on a_1 to a_4). **(b)** For the bottom-level encoding, MLEC can

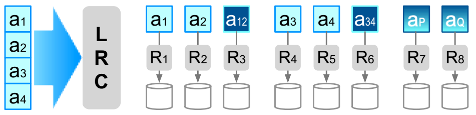

Figure 14: A (4,2,2) LRC (§5.2.1). *To be compared with MLEC layout in Figure 2c. Here, a_P and a_Q are the first and second parities of a_1 to a_4 using specific LRC encoding formulas [23].*

have multiple parities in each local stripe (*e.g.*, an ∗/(4+2) MLEC has 2 local parities in a local stripe), but LRC always has one single parity in each local group (*e.g.*, a_{12} in Figure 14). **(c)** Regarding the double parity, MLEC always computes double parities from network parities (*e.g.*, back in Figure 2.1c, a_P is based on a_{13} and a_{24}), but many LRCs don't do the same (although some of its variants do [46, 47]). **(d)** On chunk placement, MLEC puts one local stripe in a local pool on a separate rack, and can choose de/clustered placement for both intra- and inter-pool, resulting in four possible schemes. In contrast, LRC usually puts every chunk in a separate rack in a declustered way [23, 48]. Although it's possible for LRC to put one local group in the same rack [43], we are not aware of any existing LRC systems that adopt this.

5.2.2 *Durability vs. Throughput.*

Due to these differences, MLEC and LRC provide different tradeoffs in performance and durability, which we evaluate here. **Figure 15** quantifies the durability and throughput tradeoff in MLEC and LRC. Here, we only show declustered LRC ("LRC-Dp") as the most common configuration; we never found "LRC-Cp." For the MLEC scheme, we only show $^C/_D$ as it gives the best durability among all the other schemes. Just like previously, we compare various configurations all having capacity (parity space) overhead of around 30%.

Finding #1: *MLEC can provide high durability with higher encoding throughput.* This is because LRC only has one parity in each local group, and thus depends on more global parities to provide higher durability, but again more global parities can degrade the encoding throughput. Moreover, LRC-Dp places chunks in a one-level declustered way just similar to network-Dp SLEC, making it have a similar durability pattern as network-Dp SLEC.

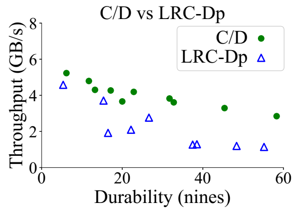

Figure 15: MLEC vs. LRC durability and throughput tradeoff (§5.2.2).

Finding #2: *The multi-level nature of MLEC allows fewer parities in each level, which then helps MLEC alleviate durability loss due to failure detection time.* Note that when multiple disks fail, a declustered (Dp) pool usually has a very small number of high-priority stripes (*i.e.*, stripes that have multiple failed chunks). Such stripes are prioritized and repaired fast, leading to high durability. This durability benefit increases with more parities and larger pool size (due to an even smaller number of high-priority stripes). *However*, because there is a 30-minute failure detection time (commonly used [23, 43, 48]), the increased durability diminishes and is not fully reflected. This is the reason why both $^C/_D$ and LRC-Dp lose some durability. But, since $^C/_D$ performs declustered parity in a local pool (which is smaller than LRC-Dp's pool) and has fewer parities at each level, the declining durability is less severe compared to LRC-Dp.

Furthermore, our multi-level-aware repair optimizations further improve $^{C}/_{D}$'s durability, helping it reach high durability with higher encoding throughput compared to LRC-Dp. We also note that if failure detection time is reduced significantly (*e.g.*, to 1 minute), LRC-Dp's durability could be similar or slightly better than MLEC, which we will explore in the future. However, such fast failure detection is *not* realistic as disks often fail transiently and should not be rebuilt hastily.

5.2.3 Failure Burst Tolerance. Just like in Section 5.1.3, we now measure the PDL of LRC under correlated failure bursts, as shown in **Figure 16**, using a (14, 2, 4) LRC-Dp. We pick this LRC configuration as it has a similar throughput rate as our (10+2)/(17+3) MLEC configuration, based on previous section's findings. However, we emphasize that their actual values in the figure should *not* be directly compared with the MLEC's PDL results shown earlier in Figure 5, as it would be an unfair comparison since one could always increase the number of parities and stripe width to get better durability/PDL. We already compare MLEC vs. LRC fairly in the previous section.

LRC-Dp is susceptible to highly scattered failure bursts. Since LRC-Dp places chunks in a one-level declustered way across racks, its failure burst tolerance pattern is similar to network-Dp SLEC (shown earlier in Figure 13d), and can lose data under highly scattered failure bursts. However, as discussed earlier in Section 4.1.1, MLEC in general is robust to highly scattered failures.

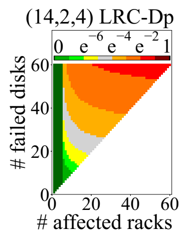

Figure 16: PDL pattern of LRC under correlated failures.

5.2.4 Repair Network Traffic. Finally, we analyze the repair network traffic of LRC-Dp. We find that LRC-Dp's repair network traffic is less than network SLEC since most failures can be repaired using the local group with less chunks. However, every repair still needs to read and write over the network, which can still lead to lots of repair network traffic. On the other hand, MLEC requires much less network traffic. We do not show any figure due to space constraints.

6 DISCUSSIONS

6.1 Takeaways

We believe our findings will provide guidance for large-scale storage architects to choose ideal configurations for their particular environments and requirements. Here are some examples:

(1) For institutions without large storage devops teams, they can buy RBODs from storage vendors, build MLEC on top easily, and choose RepairALL with some sacrifice in performance and durability.
(2) Those with more flexibility can optimize their MLEC with our advanced repair techniques such as RepairMIN.
(3) Systems detecting frequent occurrences of correlated failure bursts should utilize C/C to get better failure burst tolerance.
(4) Systems with rare failure bursts should use C/D or D/D to get higher durability under independent failures.
(5) Systems with lower durability requirements should choose SLEC for better performance.
(6) Systems that prioritize high durability (e.g. certain HPC systems where any lost chunk can make PBs of correlated data useless) should choose MLEC to minimize overheads.

We also hope our paper will encourage more research on MLEC for ML/HPC/cloud systems. For example, when offloading analytics to computational storage in HPC systems, efficiently mapping logical objects to physical blocks in erasure-coded systems poses a challenge. MLEC adds complexity to this problem due to its layering, which can be explored in future work.

6.2 Reproducibility of the Study

This project has been ongoing for over 1.5 years involving a group consisting of a theorist, a large-scale system administrator (>20k disks), a developer/maintainer, an architect, and academics. The diversity of roles ensures that the assumptions in our work closely match real-world scenarios.

The group has verified many important failure cases, possible/practical repair methods, and actual system architectures.

We together scrutinized the correctness of every scheme and method. We went back and forth to fix errors/misassumptions. Our multiple methodologies verify each other. For example, when the simulator result didn't conform with the theoretical model, the theorist and simulator developer went back and forth in multiple iterations to resolve discrepancies.

Our results are fully reproducible. We have released our source code [24] on Github and a detailed evaluation artifact [25] on Chameleon Trovi.

7 CONCLUSION

We have provided comprehensive design considerations and analysis of MLEC at scale. MLEC can be designed in multiple dimensions. We have quantified their performance and durability with various evaluation strategies, and have shown which MLEC schemes and repair methods can provide the best failure tolerance and greatly reduce repair network traffic. We have also shown that MLEC can provide high durability with higher encoding throughput and less repair network traffic over other EC schemes.

8 ACKNOWLEDGMENTS

We thank the anonymous reviewers for their tremendous feedback and comments. We also would like to thank Gary Grider from Los Alamos National Lab (LANL) for his helpful discussions on real-world deployment considerations of MLEC. This material was supported by funding from NSF grant No. CCF-2119184, funding from Oak Ridge National Laboratory (ORNL) under Contract No. DE-AC05-00OR22725 with Office of Science of the U.S. Department of Energy, as well as generous donations from Seagate. The experiments in this paper were performed in the Chameleon [49, 50] testbed.

Any opinions, findings, and conclusions, or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the NSF or other institutions.

REFERENCES

[1] D. Colarelli and D. Grunwald. Massive Arrays of Idle Disks For Storage Archives. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing (SC)*, 2002.

[2] Huaxia Xia and Andrew A. Chien. RobuSTore: Robust Performance for Distributed Storage Systems. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing (SC)*, 2007.

[3] Zizhong Chen. Optimal real number codes for fault tolerant matrix operations. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2009.

[4] Haiyang Shi and Xiaoyi Lu. TriEC: Tripartite Graph Based Erasure Coding NIC Offload. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2019.

[5] Haiyang Shi and Xiaoyi Lu. INEC: Fast and Coherent In-Network Erasure Coding. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

[6] Liangfeng Cheng, Yuchong Hu, Zhaokang Ke, Jia Xu, Qiaori Yao, Dan Feng, Weichun Wang, and Wei Chen. LogECMem: Coupling Erasure-Coded In-Memory Key-Value Stores with Parity Logging. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.

[7] Yuya Uezato. Accelerating XOR-based erasure coding using program optimization techniques. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.

[8] Salvatore Di Girolamo, Daniele De Sensi, Konstantin Taranov, Milos Malesevic, Maciej Besta, Timo Schneider, Severin Kistler, and Torsten Hoefler. Building blocks for network-accelerated distributed file systems. In *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2022.

[9] 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.

[10] Jaeho Kim, Jongmin Lee, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Enhancing SSD reliability through efficient RAID support. In *Proceedings of the Asia-Pacific Workshop on Systems (APSys)*, 2012.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] Scality ARTESCA: Object Storage for S3 Applications. https://www.scality.com/products/artesca/.

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

[18] Jehan-François Pâris, S. J. Thomas J. E. Schwarz, Ahmed Amer, and Darrell D. E. Long. Highly reliable two-dimensional RAID arrays for archival storage. In *31th IEEE – International Performance Computing and Communications Conference (IPCCC)*, 2012.

[19] Neng Wang, Yinlong Xu, Yongkun Li, and Si Wu. OI-RAID: A Two-Layer RAID Architecture towards Fast Recovery and High Reliability. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2016.

[20] Alexander Thomasian. Multi-level RAID for very large disk arrays. In *ACM SIGMETRICS Performance Evaluation Review*, 2006.

[21] 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 20st ACM Symposium on Principles of Distributed Computing (PODC)*, 2001.

[22] Alexander Thomasian and Yujie Tang. Performance, Reliability, and Performability Aspects of Hierarchical RAID. In *2011 IEEE Sixth International Conference on Networking, Architecture, and Storage (NAS)*, 2011.

[23] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. Erasure Coding in Windows Azure Storage. In *Proceedings of the 2012 USENIX Annual Technical Conference (ATC)*, 2012.

[24] MLEC Github repository. https://github.com/ucare-uchicago/mlec-sim.

[25] MLEC Artifact on Chameleon Trovi. https://tinyurl.com/mlec-artifact.

[26] 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.

[27] Mark Holland and Garth 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] CORVAULT - Self-Healing, High Density Data Storage. https://www.seagate.com/products/storage/data-storage-systems/corvault/.

[33] Jeff Bonwick and Bill Moore. Zfs: The last word in file systems, 2007.

[34] Dell PowerEdge RAID Controller 12. https://infohub.delltechnologies.com/p/dell-poweredge-raid-controller-12/.

[35] Paul Glasserman, Philip Heidelberger, Perwez Shahabuddin, and Tim Zajic. Splitting for rare event simulation: analysis of simple cases. In *Proceedings of the 28th conference on Winter simulation*, pages 302–308, 1996.

[36] Victor F Nicola, Perwez Shahabuddin, and Marvin K Nakayama. Techniques for fast simulation of models of highly dependable systems. *IEEE Transactions on Reliability*, 50(3):246–264, 2001.

[37] Daniel Ford, Franis Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlna. Availability in Globally Distributed Storage Systems. In *Proceedings of the 9th Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

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

[39] Hiroaki Akutsu and Tomohiro Kawaguchi. Reliability analysis of distributed raid with priority rebuilding. In *Proc. USENIX Conf.*, 2013.

[40] Kishor S Trivedi. *Probability and statistics with reliability, queuing, and computer science applications*. John Wiley & Sons, 2001.

[41] ORNL's Alpine storage system. https://www.olcf.ornl.gov/olcf-resources/data-visualization-resources/alpine.

[42] Personal Communication with LANL, ORNL, and Seagate Engineers and Operators.

[43] Yuchong Hu, Liangfeng Cheng, Qiaori Yao, Patrick P. C. Lee, Weichun Wang, and Wei Chen. Exploiting Combined Locality for Wide-Stripe Erasure Coding in Distributed Storage. In *Proceedings of the 19th USENIX Symposium on File and Storage Technologies (FAST)*, 2021.

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

[45] 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.

[46] Oleg Kolosov, Gala Yadgar, Matan Liram, Itzhak Tamo, and Alexander Barg. On Fault Tolerance, Locality, and Optimality in Locally Repairable Codes. In *Proceedings of the 2018 USENIX Annual Technical Conference (ATC)*, 2018.

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

[48] Saurabh Kadekodi, Shashwat Silas, David Clausen, and Arif Merchant. Practical Design Considerations for Wide Locally Recoverable Codes (LRCs). In *Proceedings of the 21th USENIX Symposium on File and Storage Technologies (FAST)*, 2023.

[49] Chameleon - A configurable experimental environment for large-scale cloud research. https://www.chameleoncloud.org.

[50] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (ATC)*, 2020.

Appendix: Artifact Description/Artifact Evaluation

ARTIFACT DOI

10.5281/zenodo.8231461

ARTIFACT IDENTIFICATION

In this paper, we provide comprehensive design considerations and analysis of multi-level erasure coding (MLEC) at scale. We introduce the design space of MLEC in multiple dimensions, including various code parameter selections, chunk placement schemes, and various repair methods. We quantify their performance and durability, and show which MLEC schemes and repair methods can provide the best tolerance against independent/correlated failures and reduce repair network traffic by orders of magnitude. To achieve this, we use various evaluation strategies including simulation, splitting, dynamic programming, and mathematical modeling. We also compare the performance and durability of MLEC with other EC schemes such as SLEC and LRC and show that MLEC can provide high durability with higher encoding throughput and less repair network traffic over both SLEC and LRC.

Our computational artifacts provide the software for evaluating MLEC and other EC schemes using the strategies mentioned above. These artifacts include:

- A simulator with “normal” mode to measure performance such as repair network traffic, and “splitting” mode to measure high durability under independent failures.
- Programs that use dynamic programming to compute the probability of data loss (PDL) under correlated failure bursts.
- A tool based on Intel ISA-L to evaluate the encoding throughput.
- Scripts for plotting the figures in the paper based on the experiment results.
- Jupyter notebooks with step-by-step guides that can run on ChameleonCloud to reproduce multiple representative figures in the paper.

These artifacts enable users to quantify the performance and durability of MLEC across various code parameters, chunk placement schemes, and repair methods. Additionally, they can also be used to evaluate other EC schemes such SLEC and LRC.

We provide step-by-step Jupyter notebooks to reproduce multiple figures in the paper including Figure 5, Figure 8, Figure 10, Figure 11, and part of Figure 12.

REPRODUCIBILITY OF EXPERIMENTS

Users can run our artifact on Chameleon Trovi: https://www.chameleoncloud.org/experiment/share/7a309c34-482b-4eac-b234-bbe4334830f2

Users can run our artifact by running the Jupyter notebooks in the following order:

- Setup.ipynb should be run first to reserve the node from ChameleonCloud, build the OS image, install the required packages, and download and set up our simulator and evaluation tools. This notebook can take 10-20 minutes.

- Fig5.ipynb computes burst tolerance for different MLEC schemes and repair methods using dynamic programming. It then reproduces Figure 5 based on experiment results. It should take 10 minutes to run. - Fig8.ipynb runs simulation in "normal" mode and evaluates the repair network traffic for different MLEC schemes and repair methods. It then plots Figure 8 based on experiment results. It should take 5-10 minutes to run.

- Fig10.ipynb runs simulation in "splitting" mode to simulate the high durability for different MLEC schemes and repair methods. It then plots Figure 10 based on experiment results. It takes 3-4 hours to finish. Since it takes a long time, we run the experiments in the background using "tmux". We also provide a script to help monitor if the experiments have finished or not.

- Fig11.ipynb measures the encoding throughput for different SLEC configurations and then plot a heatmap (Figure 11) for it. It takes 2-3 hours to finish and we also run it in the background.

- Fig12a.ipynb evaluates the durability and throughput for different EC schemes. The durability evaluation is done using the "splitting" method, which is complicated to configure and time-consuming to run. Therefore, we provide well-prepared scripts to run the experiments, and reproduce 10 data points from Figure 12a. These data points should be representative enough to show the patterns and findings mentioned in the paper. It takes 3-5 hours to finish and we also run it in the background.

In summary, the entire artifact takes 8-12 hours to run. But many of then are configured to run in the background using "tmux", which we hope can help save reviewers’ time

ARTIFACT DEPENDENCIES REQUIREMENTS

Hareware: A machine with >200 cores. Our Jupyter notebook can run to reserve "compute_zen3" node from ChanemeleonCloud automatically.

Operating system: Ubuntu 20. Our Jupyter notebook can build it on the reserved Chameleon node automatically.

Software libraries: conda, numpy, matplotlib, mpmath, pandas, yasm, nasm, ISA-L. We provide a "setup-node.sh" to build the libraries.

input datasets: none.

ARTIFACT INSTALLATION DEPLOYMENT PROCESS

Users can run our artifact on Chameleon Trovi: https://www.chameleoncloud.org/experiment/share/7a309c34-482b-4eac-b234-bbe4334830f2

You can also find our artifact by searching for "SC23 MLEC Artifact" on Chameleon Trovi: https://chameleoncloud.org/experiment/share/

You can launch it by simply clicking "Launch on Chameleon" button on the right.

We provide a Setup.ipynb with ready-to-run scripts to deploy the experiments. Deployment can take 10-20 minutes.