Practical Design Considerations for Wide Locally Recoverable Codes (LRCs)

SAURABH KADEKODI, SHASHWAT SILAS, DAVID CLAUSEN, and ARIF MERCHANT, Google

Most of the data in large-scale storage clusters is erasure coded. At exascale, optimizing erasure codes for low storage overhead, efficient reconstruction, and easy deployment is of critical importance. *Locally recoverable codes (LRCs)* have deservedly gained central importance in this field, because they can balance many of these requirements. In our work, we study wide LRCs; LRCs with large number of blocks per stripe and low storage overhead. These codes are a natural next step for practitioners to unlock higher storage savings, but they come with their own challenges. Of particular interest is their *reliability*, since wider stripes are prone to more simultaneous failures.

We conduct a practically minded analysis of several popular and novel LRCs. We find that wide LRC reliability is a subtle phenomenon that is sensitive to several design choices, some of which are overlooked by theoreticians, and others by practitioners. Based on these insights, we construct novel LRCs called *Uniform Cauchy LRCs*, which show excellent performance in simulations and a 33% improvement in reliability on unavailability events observed by a wide LRC deployed in a Google storage cluster. We also show that these codes are easy to deploy in a manner that improves their robustness to common maintenance events. Along the way, we also give a remarkably simple and novel construction of distance-optimal LRCs (other constructions are also known), which may be of interest to theory-minded readers.

CCS Concepts: • **Computer systems organization** → **Redundancy**; *Reliability;*

Additional Key Words and Phrases: Reliability, erasure codes, distributed storage systems

ACM Reference format:
Saurabh Kadekodi, Shashwat Silas, David Clausen, and Arif Merchant. 2023. Practical Design Considerations for Wide Locally Recoverable Codes (LRCs). *ACM Trans. Storage* 19, 4, Article 31 (November 2023), 26 pages.
https://doi.org/10.1145/3626198

1 INTRODUCTION

Large-scale storage clusters currently house exabytes of data, the bulk of which is encoded with erasure codes. With storage devices (hard-disk drives or just disks) routinely becoming unavailable (due to maintenance or even failures), using erasure coding of some variety is essential to provide acceptable data durability. But this durability comes with the additional storage overhead incurred by erasure codes. At a time when data corpus size is growing exponentially [10, 47], reducing this storage overhead is essential. One way to accomplish this is to utilize wider stripes for encoding, i.e., codes that have a higher ratio of data blocks to coded/redundancy blocks. Such

S. Kadekodi and S. Silas contributed equally to this research.
Authors' address: S. Kadekodi, S. Silas, D. Clausen, and A. Merchant, Google; e-mails: saukad@google.com, shashwat@alumni.stanford.edu, aamerchant@google.com, dclausen@google.com.

1553-3077/2023/11-ART31
https://doi.org/10.1145/3626198

codes are sometimes called "wide codes," since they require the overall width of the stripe to be larger (width is also referred to as *blocklength* in the coding theory literature). To unlock large storage savings without compromising reliability, codes of larger widths like 20 [4] have been deployed, and in this work, we present some data from a Google storage cluster using an erasure code of width ≈80 blocks. The drawback of most wide codes with low overhead is that they may require a large amount of IO to reconstruct any unavailable or lost data. This is why wide codes are usually designed as ***locally recoverable codes (LRCs)*** (Definition 5.3), which help mitigate this reconstruction cost in most cases. Wide LRCs can be utilized to balance the challenging storage needs of today (low storage overhead, competitive reliability, competitive average reconstruction cost), but unique challenges arise when designing *wide* LRCs for deployment in storage clusters.

LRCs are optimized to shine in the case when there is a single erasure in a stripe, but much effort has gone into designing LRCs with various other desirable properties [19, 28, 29, 35, 51]. One obvious direction that has received much attention is to design *distance-optimal LRCs* [35, 53]—that is, LRCs with the best possible *distance*, given their width, dimension, and locality. A code with distance d guarantees recovery from any pattern of up to $d-1$ failures. Indeed, maximizing distance is especially important for wide codes, since they are more likely (simply due to their width) to encounter a larger number of failures simultaneously, as we show using data from Google storage clusters in Figure 2. But, we find that even if we use distance-optimal LRCs, storage clusters using wide codes encounter a meaningful number of events where the number of failures is *larger* than the distance of the distance-optimal LRC (see data from a Google storage cluster in Figure 1). So, it becomes important to study LRCs that can successfully reconstruct even a significant fraction of erasure patterns *beyond the optimal distance*. The theory community has been tackling this problem by studying *maximally recoverable locally recoverable codes (MR-LRCs)*, which take the erasure correction capability of an LRC to the information theoretic limit implied by its design parameters [18, 20, 22]. However, to the best of our knowledge, current constructions of MR-LRCs do not yield codes fitting the limitations of current hardware. For example, wide MR-LRCs with storage overhead <20% require orders of magnitude larger field sizes than the computationally efficient field sizes of up to $\mathbb{F}_{256}$, which is explained in detail in Section 4.

Optimizing LRC reliability in practical parameter settings is a valuable and open problem, and we tackle aspects of it in this work. We highlight some of our main contributions.

Practical measurement of reliability. We study wide LRCs with a distinctively practical lens. One of our contributions is the curation of a set of *robust and practical* measures of LRC reliability. These include performance against random erasures, a comparison of the reliability of explicit LRC generator matrices (see Definition 5.1) against the information theoretic limit provided by MR-LRCs, and calculations of ***mean time to data loss (MTTDL)*** using *observed* reliability metrics. We also provide a careful study of how random erasures can affect the reliability not only of a single stripe, but of stripes at a system scale. Finally, we develop the notion of a *maintenance zone* to show that the precise deployment of a coding scheme in a data center can also affect the performance. As mentioned earlier, going wide creates new reliability challenges for LRCs, and these tools provide a clearer and more realistic set of tests to tackle the new challenges. Using these tools, we meaningfully compare popular deployed LRCs and showcase novel highly performant LRCs.

New distance-optimal LRC constructions. In Section 6, we provide a novel construction of distance-optimal LRCs (Definition 5.4), which we call *Optimal Cauchy LRCs*. This continues a long line of work [51, 53] for constructing distance-optimal LRCs, and while our construction is not the most general, it is remarkably simple and yields many codes (even wide LRCs) in practically useful parameter settings.

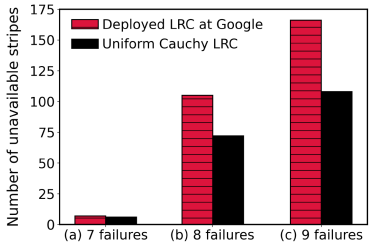


Fig. 1. We captured a sample of 278 unavailable stripes captured from four Google storage clusters, along with information about the exact block failures in each sample. The deployed code had total width $n \approx 50$ and always succeeded in recovering data when there were ≤ 6 failures. We then test these failure scenarios with the Uniform Cauchy LRC of the same width and overhead. The deployed code could not recover any of the 278 stripes before restoration, whereas Uniform Cauchy LRC simulation was successful in recovering 92 stripes prior to restoration; a success ratio of 33%.

Insights into reliable wide LRC design. Our practical measures of reliability provide several insights. While it is true that higher distance gives guarantees about fixing up to a certain number of erasures, it is not enough by itself to guarantee strong empirical results. The first improvement that can be made in this direction is to find codes that are *approximately MR-LRCs* (since it is not yet possible to construct true MR-LRCs in our parameter regimes). Here, we show good news: The coefficients used in our simple constructions get us over 99% of the reliability possible with MR-LRCs. But, we find that just being (close to) maximally recoverable is not the end of the journey for reliability. Indeed, two codes that have the same width and same storage overhead and are both MR-LRCs (true or approximate) can have significantly different resilience to *random patterns of erasures*. This is because erasure recovery is affected by how failures are distributed across the local repair groups (see Definition 5.5) of the LRC, so it is not enough to just optimize the coefficients of an LRC, but also the *size* of its local repair groups. To the best of our knowledge, this fact has not been considered in the literature, even though our experiments show that it can have a significant impact on reliability. In general, codes whose local groups are *evenly sized* have better performance (see discussion in Section 8). This has the additional perk of also lowering the degraded mode read cost and the reconstruction cost of the code.

Novel LRCs that excel in practice. Using some of these insights, we modify our construction of distance-optimal codes to create *Uniform Cauchy LRCs*. We find that these codes truly shine in most practically relevant reliability (and performance!) measures. Figure 1 shows data unavailability events from a deployed wide LRC of width ≈50 blocks along with their erasure patterns captured from four large storage clusters at Google with a total disk population of over 1.7 million disks, over a period of one year. For the same code width and storage overhead, our Uniform Cauchy LRC simulation recovered more than 33% of these stripes without the need for nodes to become available again. Indeed, further experiments confirm this observation by showing that Uniform Cauchy LRCs outperform many popular (and deployed) LRCs across our metrics. A comprehensive experimental evaluation of LRCs is provided in Section 8, along with the main observations.

Maintenance-robust deployment of wide LRCs. In Section 9, we highlight the importance of maintenance-robust deployment of wide LRCs. Even though a code may have many desirable properties, its exact layout in a cluster affects its robustness to common maintenance events such as kernel upgrades. It is desirable to construct codes that are easier to deploy in a maintenance-robust manner (not all codes are equal here), and we quantitatively show that the design of Uniform Cauchy LRCs is beneficial in this regard.

Our work shows that myriad design choices need to be considered to optimize wide LRCs. Indeed, accounting for these factors can lead to more reliable deployments of wide LRCs in practice.

2 BACKGROUND

Large-scale storage clusters. Large-scale storage clusters typically consist of public cloud offerings or high-performance computing systems. **Hard-disk drives (HDDs)** make up the primary storage tier in these clusters. It is common for the disk population in a large-scale cluster to be above 100k [3, 34], and some large ones are also reported to have close to 500k disks [32, 33]. Data being stored in large-scale storage clusters is increasing at an alarming rate [8, 10, 11, 45]. Data redundancy using erasure coding is the reliability mechanism of choice for bulk of the data.

Erasure coding. Erasure coding is a more space-efficient alternative to data replication. Usually described as an (n, k) code, an erasure coding *stripe* encodes k data blocks (typically one or a few megabytes in size) along with $n - k$ "parity" blocks (of the same size) to form an n block *stripe*. The storage overhead is calculated as $\frac{n}{k}$. **Maximum Distance Separable (MDS)** codes (like Reed-Solomon codes) are popular erasure codes used in practice, because they provide the maximum erasure correction capability for any fixed value of n and k. MDS codes have the property that up to $n-k$ blocks missing from a stripe can be reconstructed using *any* k of the remaining blocks. As data has grown and space-efficiency has become more critical, *wider* MDS codes (i.e., codes with larger values of k) with less storage overhead have become more popular in practice. Indeed, even (20, 17) MDS codes have been deployed and studied [4], in place of the once-ubiquitous schemes like RAID-6 (which is a (6, 4)-Reed-Solomon code), (9, 6) [15], and (14, 10) [46]. But wider MDS codes have their own drawbacks, because they require all the data from k blocks to reconstruct/repair even a single missing block, leading to a high *reconstruction cost*. A need to use wider encoding schemes with less overhead, combined with the very high reconstruction cost of MDS codes has motivated the study of Locally Recoverable Codes.

Data reconstruction process. Large-scale cluster storage systems have a background process that monitors the redundancy level of all stripes stored in the cluster. Whenever a disk becomes unavailable (either due to server unavailability or disk failure), the background daemon flags the under-redundant stripe and starts a timeout of a few tens-of-minutes. When the timeout expires, the stripe is marked for reconstruction. Storage clusters often set a soft threshold on the bandwidth used for background activity such as reconstructions (except client-initiated degraded mode reads). To balance the reconstruction workload while maintaining highest data safety standards, the reconstructions are processed via a priority queue in which stripes that are more vulnerable are reconstructed before less-vulnerable stripes.

Locally Recoverable Codes (LRCs). LRCs [19, 29, 35, 46, 51, 53] (also known as Local Reconstruction Codes) are erasure codes designed to mitigate the high reconstruction cost of MDS codes. An (n, k, ℓ) LRC code divides an n block stripe into local groups, each with at most $\ell < k$ blocks and a local parity.[1] In addition to the local parities, the stripe also has *global parities* that cover all k data blocks. One may think of a *typical* LRC in this way: The data blocks and the global parities together form an MDS code, and the local parities are added on top of this code to mitigate the cost of reconstruction (reduce it from k to ℓ) in the case when there is *exactly* one failure in a local group. One may note that LRCs are not MDS codes, since they cannot satisfy the Singleton bound unless $\ell = k$ (see Definition 5.6). If there is more than one failure in the same local group, then the underlying MDS code can be used to reconstruct the data. Several different LRC constructions have been proposed over the years with different trade offs [29, 35, 51, 53].

[1]Theoretically, each local group can have any number of parities, but in practice the most common configuration involves 1 parity per local group.

Distance of a code. Distance of a code (denoted by d) is the minimum number of failures/erasures that may render the stripe potentially non-recoverable (i.e., all patterns of $< d$ failures are always recoverable). For example, the distance of an (n, k) MDS code would be $n - k + 1$, since an MDS code can recover from any $n - k$ failures. In fact, for an MDS code, any failure beyond $n - k$ failures is strictly non-recoverable. However, for a code with distance d that is *not* MDS (such as an LRC), some patterns of $\geq d$ failures could be recovered. Maximizing this capability to recover as many erasure patterns as possible *beyond the distance* leads us to the notion of *maximally recoverable codes*.

Maximally Recoverable LRCs (MR-LRC). For any code, simply specifying whether each entry in its generator matrix (see Definition 5.1) is zero or non-zero imposes an information theoretic limit on which patterns of erasures could be recoverable. The study of *maximally recoverable codes* is concerned with finding coefficients for the non-zero entries such that this limit is reached [18]. LRCs can also be optimized in this way: Once we have specified which entries of the generator matrix are non-zero (this will fix various code parameters such as n, k, number of local parities, number of global parities, and the size of the local groups), it is possible to find coefficients that maximize the number of recoverable erasure patterns (including potentially many patterns of $\geq d$ erasures). LRCs that are *maximally recoverable* are known as MR-LRCs. In a large-scale storage system, choosing coefficients that result in codes that are (close to) MR-LRCs is a way of ensuring the highest possible data availability.

3 MOTIVATION FOR STUDYING WIDE LRCS

Reducing storage overhead is critical. Storage overhead because of data redundancy is a major component of a cluster's storage cost. With 3-way replication, the overhead is 3×, which is prohibitive for large-scale clusters storing **exabytes (EBs)** of data on hundreds-of-thousands of disks. Even the popular MDS codes such as $(9, 6)$ [15] or $(14, 10)$ [46], which have an overhead of 1.5× and 1.4×, respectively, are considered too expensive for exascale [10, 11, 31, 45]. Large-scale storage clusters are actively adopting *wide MDS codes* to minimize the storage overhead. Backblaze reported the use of a $(20, 17)$ MDS code [4], which has a reasonably low overhead of 1.17× (a rate of $\frac{17}{20} = 0.85$; see Definition 5.1). With every percent reduction in storage overhead resulting in savings of millions in capital, operational and energy costs, lowering storage overhead continues to be a lucrative problem.

Wide MDS codes are costly. Using wide MDS codes has several challenges. The reconstruction IO cost of a wide (n, k) MDS code scales linearly with k. For example, while a $(9, 6)$ MDS codes requires reading 6 data blocks for reconstructing a missing block, a wide MDS code such as $(20, 17)$ requires reading 17 data blocks. Wide MDS codes also have a higher unavailability, because they have a higher probability of having blocks stored on devices that are unreachable due to maintenance (since no two blocks of a stripe can be on the same disk, server, rack, etc.). Thus, not only does a wide MDS code have a higher degraded mode read cost (Definition 5.10), but the frequency of performing degraded-mode reads is also higher due to higher unavailability. Due to a larger number of disks, degraded reads and reconstructions in wide MDS codes are also more likely to suffer from high tail latency due to stragglers. So, although wide MDS code are good for durability and reduced storage overhead, they are not good for reconstruction costs, availability, and degraded read performance, all of which are major performance concerns in large-scale storage clusters. Storage overhead minimization is critical but cannot come at the expense of aforementioned problems. Indeed, this is the reason for the continued popularity of LRCs, which mitigate this drawback.

Real-world failure patterns favor LRCs. We empirically observe the failure patterns of stripes stored in three large-scale storage clusters totalling over 1.5 million disks for a period of

6 months. These clusters have multiple different erasure coding and replication schemes deployed simultaneously and have over 1.2 trillion stripes. From among the stripes that have >0 failures, we observe that ≈99.2% of stripes have just a single failure. Similar data has been observed by others in Reference [44], where they found that ≈98.08% stripes had a single failure on a Facebook warehouse cluster. A single block failure in a stripe is a scenario where LRCs shine (in contrast to MDS codes). However, this comes at a cost of higher storage overhead (as we mentioned, typical LRCs are just MDS codes with the additional overhead of local parities, and they cannot lie on the Singleton bound).

Wide LRCs. In an ideal world, we would like to use *wide* LRCs that have significantly lower overhead than the LRCs showcased in References [35, 46] (and many other works). Wide LRCs could reduce the overhead from the 30%–60% range (which is common in deployed LRCs and MDS codes) to the less-than-20% range, while still maintaining many advantages over MDS codes. But wide LRCs create novel challenges. For example, wider stripes are much more likely to result in a larger percentage of stripes with more than a single failure. This makes it critical to study *robust and practical* measures of LRC reliability if we are to utilize wide codes in practice. Further, practical issues such as deployment and ease-of-use also need to be addressed.

4 PRACTICAL CHALLENGES OF WIDE LRCS

Many simultaneous failures are more common in wide LRCs. While most stripes (which have failures) have single-block failures, this does not mean that the tiny fraction of more than one block failure can be ignored. Moving to wider stripes increases the possibility that multiple blocks of a stripe need to be repaired at the same time, since there is a higher chance that devices or servers storing multiple blocks of the same stripe may be undergoing maintenance simultaneously. This is consistent with the observation that most data unavailability events in large-scale storage clusters are as a result of planned outages [15]. Another reason for increased number of block failures in a wide stripe is due to prioritization of reconstructions, as explained in Section 2. Figure 2 shows the number of stripes that have at least 4 failures throughout a 24-hour period. There are two LRCs whose stripe failures are being compared, one with width approximately 50 blocks and the other with width approximately 80 blocks.[2] Note that this trace is collected from a single storage cluster, and so the failures seen in this trace are in response to the same machines failing. As is clearly visible, the wider LRC has a significantly higher number of stripes with at least 4 failures compared to the relatively narrower LRC.

Constructing MR-LRCs is hard. As mentioned above, a lot of research has looked at MR-LRCs in recent years [6, 13, 18, 20, 22]. However, even with recent advancements such as References [13, 20], to the best of our knowledge, it is not possible to construct MR-LRCs in the following regime: where the field size (search-space for coefficients to construct the LRC) is fixed to be 256, n is between 25–150, and the rate of the code (see Definition 5.1) is at least 0.85 (i.e., storage overhead is at most 1.17× and $\ell < k$). This setting is particularly useful for applications and is the natural next phase for practical LRCs, which reduce overhead and maximize reliability.

Typically, maximally recoverable codes are easier to construct when the field is large, and much research is focused on finding explicit codes over small field sizes. In practical settings, most computations are done over individual bytes, which restricts the field size of the LRCs we can use to 256 (the field $\mathbb{F}_{2^8}$). To the best of our knowledge, the current state-of-the-art constructions are in Reference [20], and they require that the field size q be at least $\sim (\ell+1)^r$, where $\ell+1$ is the size of a local repair group (Definition 5.5), and r is the number of global parity checks. For our purposes,

[2]We cannot disclose the exact configuration due to confidentiality.

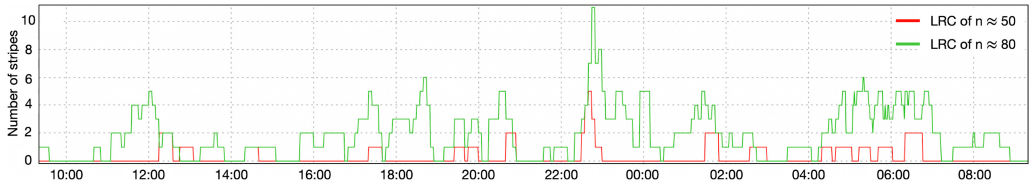


Fig. 2. A 24-hour trace with the number of stripes with at least 4 failures in two deployed LRCs captured from a single storage cluster at Google. The wider LRC ($n \approx 80$) has many more stripes with at least 4 failures compared to the relatively narrower LRC ($n \approx 50$).

we can think of $\ell + 1 \sim 16$, and $r \geq 3$, thus $q = 4,096$, making the required field size 16× greater than 256.

Measurements of reliability of wide LRCs are inadequate.

While narrow LRCs have received a lot of attention in systems research, wide LRCs have not [27]. Existing systems research on LRCs optimizes metrics such as distance, degraded read cost, and reconstruction cost [7, 29, 35, 46, 51, 53]. We enhance this rich literature with a larger suite of *empirical* measures, which give more realistic comparisons in the practical realm, particularly in the case when there are $\geq d$ erasures in a code with distance d. Further, since explicit MR-LRCs are not available in our parameter regimes, it leaves open the question of evaluating whether we can construct codes that offer us *approximately* the same advantages as MR-LRCs could offer (the answer is yes!). Finally, some design choices such as creating evenly sized local groups have not been considered in the literature at all, which we show can have a large impact on performance against random patterns of erasures.

Deployment of wide erasure codes is non-trivial. Another practical hurdle that gets worse with code width is the deployment (placement of blocks) of an erasure coded stripe. Recall that no two blocks of an erasure coded stripe can reside on the same disk, rack, fault-domain, power-source, and so on. As the code width increases, it becomes progressively harder to fulfill these placement constraints. The placement problem is further exacerbated because it is common for sets of servers/racks (which comprise a *maintenance zone*) to be down at the same time for maintenance events, potentially causing data unavailability. For example, if too many blocks of a stripe are in the same maintenance zone, then even a planned maintenance event could make the stripe unavailable. We comment on some ways that code design can make it easier to achieve maintenance-robust layouts of data.

5 DEFINITIONS

We begin by defining important terms that are going to be useful when describing the wide LRC construction in Section 6. These definitions and the proofs in Section 6 are somewhat formal and may be skipped by the more practically minded readers who may go directly to Section 7 after reading the necessary concepts explained through an example in Figure 3.

Definition 5.1 (Linear error correcting code, generator matrix, rate, dimension, blocklength). A linear error-correcting code is simply a subspace $C \leq \mathbb{F}_q^n$ where q is a prime power and $n > 0$. It is customary to think of C as the image of an encoding map Enc: $\mathbb{F}_q^k \to \mathbb{F}_q^n$ for some $k \leq n$. This encoding may be expressed in matrix form as,

$$Gx = y,$$

where G is an $n \times k$ matrix called the generator matrix, x is the message, and y is the codeword. The fraction $\frac{k}{n}$ is the rate of the code, k is the dimension, and n is the blocklength. The symbols $y_i \in \mathbb{F}_q$ of a codeword y are called codeword symbols.

Definition 5.2 (Distance of a code). The minimum distance of a linear code (often just called the distance) is simply,

$$\min_{x \in C \setminus \{0\}} wt(x),$$

where $wt(x) = \sum_{i=1}^{n} 1_{x_i \neq 0}$. An error correcting code with distance d can always correct $d - 1$ erasures.

Definition 5.3 ((n, k, ℓ)-Locally recoverable code (LRC)). An (n, k, ℓ)-LRC is a linear error correcting code of dimension k and blocklength n. It has the additional property that any codeword symbol can be recovered from at most ℓ other codeword symbols. The parameter ℓ is called the locality paramter of the LRC. Note that $1 \leq \ell \leq k$.

Definition 5.4 (Generalized Singleton bound and distance-optimal LRCs). Any locally recoverable code must satisfy the following bound [19]:

$$n \geq k + \left\lceil \frac{k}{\ell} \right\rceil + d - 2,$$

where d is the distance of the code. Any LRC that meets this bound with equality is called a distance-optimal LRC. Note that when $\ell = k$, this reduces to the well-known Singleton bound $n \geq k + d - 1$.

Definition 5.5 (Local repair group). Given an (n, k, ℓ)-LRC, choose any codeword y. Due to the local recovery property, any codeword symbol y_i can be recovered using at most ℓ other codeword symbols $\{y_{r_1}, \ldots, y_{r_\ell}\}$ where $r_j \neq i$ for any j. These at most $\ell + 1$ indices in $\{i, r_1, \ldots, r_\ell\}$ form a local repair group of the (n, k, ℓ)-LRC.

Definition 5.6 (Maximum distance separable code). Any linear error-correcting code that satisfies the Singleton bound with equality, i.e., satisfies $n = k + d - 1$, is known as a **Maximum Distance Separable (MDS)** code.

Definition 5.7 (Cauchy matrix). Let $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_m\}$ be two disjoint sequences of distinct elements from $\mathbb{F}_q$. The $n \times m$ matrix defined as

$$C_{ij} = \frac{1}{x_i + y_j}$$

is a Cauchy matrix. It is well-known that every square submatrix of a Cauchy matrix has non-zero determinant (and, therefore, is invertible).

Fact 5.8 (Generator matrix of an MDS code). Cauchy matrices are commonly used to design generator matrices for MDS codes. Indeed, an $n \times k$ matrix over $\mathbb{F}_q$, whose first k rows form I_k (the $k \times k$ identity matrix) and whose last $n - k$ form an $(n - k) \times k$ Cauchy matrix generates an MDS code of blocklength n and dimension k (see Theorems 2.2 and 5.2 in Reference [9]).

We can visualize several of our definitions using the example shown in Figure 3.

*Definition 5.9 (**Maximally recoverable locally recoverable code (MR-LRC)**).* An (n, k, ℓ)-LRC is called an MR-LRC if it can recover from any pattern of $n - k$ erasures as long as there is at least one erasure in each local repair group of the code. To clarify, suppose the (n, k, ℓ)-LRC has local repair groups $L_1, \ldots, L_p$ for some $p > 1$ where each $L_i \subseteq \{1, \ldots, n\}$. Recall that $\cup_i L_i = \{1, \ldots, n\}$ and that the L_i are not necessarily disjoint. Let $E = \{e_1, \ldots, e_{n-k}\} \subseteq \{1, \ldots, n\}$ be any pattern of $n - k$ erased symbols. Then, as long as $L_i \cup E \neq \emptyset$ for any i, the erasure pattern E can be recovered by the code.

$$G_C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ c_{11}+c_{21} & c_{12}+c_{22} & c_{13}+c_{23} & c_{14}+c_{24} \end{pmatrix} \begin{matrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ rs_1 \\ rs_2 \\ c_1 \\ c_2 \\ c_3 \end{matrix}$$

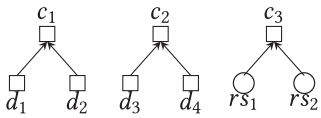


Fig. 3. Example of an LRC with 4 data blocks (d_1, d_2, d_3, d_4), 2 global parities (rs_1, rs_2), and 3 local parities (c_1, c_2, c_3). This figure shows how each row of the generator matrix correponds to a "block" in the picture representation of an LRC (the reader may note that the matrix representation is more informative, as the specific choice of coefficients can affect reliability in practice). The first four rows of the matrix simply copy the data blocks to the encoding, the next two rows correspond to the global parities (these two rows form a 2×4 Cauchy matrix in our implementation); the last three rows correspond to 3 local parities, all of which contain the XOR of the data they cover (as is evident by the coefficients of the matrix G_C). The last local parity, l_g, only covers the global parities. This design of LRC may be familiar to some readers as an Azure-LRC+1. The *local groups* of this code are (d_1, d_2, c_1), (d_3, d_4, c_2), and (rs_1, rs_2, c_3) and therefore $\ell = 2$. In this code, *ADRC* (see Definition 5.10) = ARC_1 (see Definition 5.11) = 2, since each data or parity block can be reconstructed by reading the two other blocks in its local repair group.

*Definition 5.10 (**Average degraded read cost (ADRC)**).* The ADRC [35] is the average of the cost of reconstructing any of the data blocks. We can measure $cost(b_i)$ as the number of blocks that need to be read to reconstruct block i.

$$ADRC = \frac{\sum_{i=1}^{k} cost(b_i)}{k}$$

*Definition 5.11 (**Average repair (or reconstruction) cost (ARC)**).* The ARC [29, 35] is defined identically to the ARDC, with the addition of the global and local parity blocks to the computation.

$$ARC_1 = \frac{\sum_{i=1}^{n} cost(b_i)}{n}$$

For MDS codes, $\forall b_i, cost(b_i) = ARC = k$. For LRCs, $cost(b_i)$ may not be the same as ARC. The above ARC is defined for one block failing in a stripe. In our evaluation, we also show the ARC of reconstructing two failed blocks defined as

$$ARC_2 = \frac{\sum_{i=1, j\neq i}^{n} cost(b_{i,j})}{\binom{n}{2}},$$

where $1 \leq i \leq n, 1 \leq j \leq n$ and $i \neq j$, and $cost(b_{i,j})$ = cost to reconstruct blocks i and j, which can result in a combination of local and global reconstructions depending on i and j.

6 (n, k, r, p)-OPTIMAL CAUCHY LRCS

In this section, we construct (n, k, r, p)-*Optimal Cauchy LRCs* and discuss their advantages. Here, n is the blocklength of the code, k is the dimension, r is the number of global parity checks, and p is

the number of local parity checks. We make two design choices that are worth pointing out. First, the p local parities in our code are uniformly distributed across the k data, and they are all XOR-ed with the r global parity checks (this helps with proving the distance optimality of the code). Second, we restrict ourselves to the case where each of the data symbols of the code is covered by exactly *one* local parity. The second restriction is common in constructions proposed in the literature, and indeed, important for our goal of minimizing the storage overhead. For simplicity of exposition, we assume that p is even (we will mention how this condition can be removed), and $p|k$ where we denote $\frac{k}{p} = t$. It will be clear that the *locality* parameter ℓ of the (n, k, r, p)-Optimal Cauchy LRCs we will construct is $\frac{k}{p} + r$.

In Section 6.2 will show that (n, k, r, p)-Optimal Cauchy LRCs have the best possible distance for LRCs with their dimension and locality (under some modest restrictions). In Section 6.3, we show how some of these restrictions can be relaxed. We begin with explaining the construction of (n, k, r, p)-Optimal Cauchy LRCs.

6.1 Code Construction

The generator matrix for an (n, k, r, p)-Optimal Cauchy LRC is derived naturally from the generator matrix for an $(k + r + 1, k)$-MDS code, which has dimension k and blocklength $n = k + r + 1$. Specifically, we derive our code from the generator matrix of a $(k + r + 1, k)$-Cauchy MDS code.

The generator matrix $G_{(k+r+1),k}$ of a $(k + r + 1, k)$-Cauchy MDS code (see Fact 5.8) is an $k + r + 1 \times k$ matrix, which simply consists of a $k \times k$ identity matrix stacked on top of a $r + 1 \times k$ Cauchy matrix. The resulting matrix is well-known to generate an MDS code with distance $r + 2$ (see Theorems 2.2 and 5.2 in Reference [9]).

$$G_{(k+r+1),k} = \left[\begin{array}{ccc} 1 & & \\ & \ddots & \\ & & 0 \\ & \ddots & \\ 0 & & \\ & & \ddots \\ & & 1 \\ \hline c_{11} & \cdots & c_{1k} \\ & \ddots & \\ c_{r1} & & c_{rk} \\ c_{(r+1)1} & & c_{(r+1)k} \end{array}\right]$$

When we do not need to refer to r and k specifically, we refer to $G_{(k+r+1),k}$ as G and the ith row of $G_{(k+r+1),k}$ as g_i.

To create the generator matrix for an (n, k, r, p)-Optimal Cauchy LRC, we first partition this last row of G, g_{k+r+1} into p rows $r_1, \ldots, r_p$ as follows:

$$\begin{aligned} r_1 &= (c_{(r+1)1}, c_{(r+1)2}, \ldots, c_{(r+1)t}, 0, \ldots, 0) \\ r_2 &= (0, \ldots, 0, c_{(r+1)(t+1)}, \ldots, c_{(r+1)2t}, 0, \ldots, 0) \\ &\vdots \\ r_p &= (0, \ldots, 0, c_{(r+1)(p(t-1))}, \ldots, c_{(r+1)pt}). \end{aligned}$$

It is clear that $r_1 + r_2 + \cdots + r_p = g_{k+r+1}$. We then compute the rows $\tilde{r}_i$ as follows:

$$\tilde{r}_i = r_i + g_{k+1} + g_{k+2} + \cdots + g_{k+r}.$$

Note that $\tilde{r}_i$ are simply r_i with the addition of the first r "Cauchy rows" of G. Since p is even (and the underlying field has characteristic 2), we have,

$$\begin{aligned}\tilde{r}_1 + \tilde{r}_2 + \cdots + \tilde{r}_p &= p(g_{k+1} + g_{k+2} + \cdots + g_{k+r}) + g_{k+r+1} \\ &= g_{k+r+1}.\end{aligned}$$

Finally, the generator matrix $O_{n,k,r,p}$ for the (n,k,r,p)-Optimal Cauchy LRC is given by,

$$O_{n,k,r,p} = \left[\begin{array}{ccc} 1 & & \\ & \ddots & 0 \\ & & \ddots \\ 0 & & \ddots \\ & & 1 \\ \hline c_{11} & \cdots & c_{1k} \\ & \ddots & \\ c_{r1} & & c_{rk} \\ & \tilde{r}_1 & \\ & \vdots & \\ & \tilde{r}_p & \end{array}\right].$$

This matrix is simply the $k + r$ rows of $G_{k+r+1,k}$ stacked on top of the $\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_p$ vectors. We remark that $n = k + r + p$, and that $O_{n,k,r,p}$ generates an LRC with locality parameter $\ell = \frac{k}{p} + r$. This is clear, since each row among $\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_p$ provides a local parity check on exactly ℓ other rows.

6.2 Distance

We note that the distance of an (n,k,r,p)-Optimal Cauchy LRC is exactly $r+2$. As long as $k, r, p > 0$, $p|k$, $rp^2 > k + rp$, and p is even, it is a distance-optimal LRC (Definition 5.4).

Theorem 6.1. *For $r > 0$, $O_{n,k,r,p}$ generates an error correcting code with distance exactly $r + 2$.*

Proof. We will simply show that the distance of the code generated by $O_{n,k,r,p}$ is equal to the distance of the code generated by $G_{(k+r+1),k}$. Recall that $G_{(k+r+1),k}$ generates a code with distance exactly $r + 2$, which is equivalent to saying that this code can correct *any* pattern of $r + 1$ erasures, which in matrix terms means that $G_{(k+r+1),k}$ has at least k linearly independent rows (i.e., has row rank at least k) if any $r + 1$ of its rows are replaced with the zero vector.

We will show that $O_{n,k,r,p}$ has row rank at least k whenever any $r + 1$ rows are deleted. Note that whenever some $r + 1$ rows are deleted from $O_{n,k,r,p}$, they may or may not contain any rows among the $\tilde{r}_1, \ldots, \tilde{r}_p$.

First, we consider the case when at least one of $\tilde{r}_1, \ldots, \tilde{r}_p$ is deleted. In this case, we may consider *all* of them lost and note that we have exactly the same rows remaining as in g_{r+k+1}, and at most r other rows were lost in $G_{(k+r+1),k}$. Since $G_{(k+r+1),k}$ generates an MDS code with distance $r + 2$, it is clear that the remainder of the rows will have rank k.

Now consider the case when the $r + 1$ deleted rows do not contain any of the rows $\tilde{r}_1, \tilde{r}_2, \ldots, \tilde{r}_p$. In this case, we can compute $g_{k+r+1} = \tilde{r}_1 + \tilde{r}_2 + \cdots + \tilde{r}_p$, which reduces to the case when $r + 1$ rows are deleted from $G_{(k+r+1),k}$. □

Lemma 6.2. *Given $k, r, p > 0$, $rp^2 > k + rp$, and p even, the locally recoverable code formed by $O_{n,k,r,p}$ is distance-optimal.*

Proof. Recall that for a LRC to be distance-optimal, we must have,

$$n = k + \left\lceil \frac{k}{\ell} \right\rceil + d - 2, \tag{1}$$

where ℓ is the locality parameter and d is the distance. We know that $\ell = \frac{k}{p} + r$ and $n = k + r + p$ for an (n, k, r, p)-Optimal Cauchy code. Substituting ℓ and $d = r + 2$ (from Theorem 6.1), Equation (1) becomes,

$$n = k + \left\lceil \frac{k}{\frac{k}{p} + r} \right\rceil + r = k + \left\lceil \frac{pk}{k + pr} \right\rceil + r.$$

So, an (n, k, r, p)-Optimal Cauchy code will be distance-optimal whenever $\left\lceil \frac{pk}{k+pr} \right\rceil = p$, i.e., $p-1 < \frac{pk}{k+pr} \leq p$. Both inequalities hold as long as $r, p, k > 0$ and $rp^2 < k + rp$. □

We now show that some the conditions in Lemma 6.2 can be relaxed.

6.3 Relaxing Constraints

We have shown how to construct distance-optimal LRCs with blocklength $n = k + r + p$, where k is the dimension of the code, r is the number of global parity checks, and p is the number of local parity checks as long as p is even, $r, p, k > 0$ and $rp^2 < k + rp$. We will briefly discuss some extensions of these parameter regimes in this section.

Claim 6.3. *We may modify our construction of distance-optimal codes to work with p odd.*

If $p = 1$, then we may modify the construction so $r_1 = r_p = \tilde{r}_p = g_{k+r+1}$, and so $G_{(k+r+1),k} = O_{n,k,r,p}$. This just reduces to the code defined by $G_{(k+r+1),k}$.

If $p \geq 3$, then we may modify the construction so $\tilde{r}_i = r_i$ for $1 \leq i \leq p-2$, $\tilde{r}_{p-1} = r_{p-1} + g_{k+1} + \cdots + g_{k+r}$, and $\tilde{r}_p = r_p + g_{k+1} + \cdots + g_{k+r}$. In this case,

$$\begin{aligned} \tilde{r}_1 + \tilde{r}_2 + \cdots + \tilde{r}_p &= 2(g_{k+1} + g_{k+2} + \cdots + g_{k+r}) + g_{k+r+1} \\ &= g_{k+r+1}. \end{aligned}$$

From here, the same reasoning as the previous sections implies that these codes are distance-optimal. Notice that when $p \geq 3$, these modifications do not change ℓ, because $\tilde{r}_p$ is a local parity check on $\frac{k}{p} + r$ data blocks.

Claim 6.4. *If $\frac{rp^2}{2} < k + rp < rp^2$, then our construction gives codes whose blocklength is at most one greater than a distance-optimal code.*

This is easily observed by following the proof of Lemma 6.2 and considering the cases where $p - 2 < \frac{pk}{k+pr} \leq p - 1$. Previous works have also found such "off by one" LRCs in broader parameter regimes than truly optimal LRCs [51]. This allows us more freedom to explicitly construct almost distance-optimal and wide LRCs.

Claim 6.5. *We may construct codes with alphabet size $q \geq k + r + 1$.*

Note that we can construct our LRCs starting from the generator matrix of any MDS code in place of $G_{k+r+1,k}$. In particular, taking $G_{k+r+1,k}$ to be the generator matrix of any Reed-Solomon code, we only need $q \geq k + r + 1$ (since Reed-Solomon codes need $q \geq n$). For our settings, q is fixed to be 256, so we have the flexibility to construct many wide explicit codes.

7 (n, k, r, p)-UNIFORM CAUCHY LRCS

In the previous section, we give a very simple construction of distance-optimal LRCs that we will use in our experimental analysis as examples of distance-optimal codes. In this section, we provide a simple heuristic modification of these codes that has several practical advantages (though they are not shown to be optimal with regards to distance).

The later experimental analysis of these codes highlights the point that distance-optimal (i.e., on the generalized Singleton bound) does not mean most-durable or most cost-efficient from a practical perspective. For example, a code that has the same locality and distance can have different durability (as measured by mean-time-to-data-loss) and robustness against random patterns of erasures. Distance-optimality simply indicates the best distance for fixed values of n, k, and ℓ.

These codes are constructed in much the same way as Optimal Cauchy LRCs, except that each local parity check covers $\frac{k+r}{p}$ of the data blocks and global parity blocks. Following the same notation as the previous section, the generator matrix for a uniform Cauchy code has the form (assuming $p|k+r$ for simplicity),

$$U_{n,k,r,p} = \left[\begin{array}{ccc} 1 & & \\ & \ddots & 0 \\ & & \ddots \\ 0 & & \ddots \\ & & 1 \\ \hline c_{11} & \cdots & c_{1k} \\ & \ddots & \\ c_{r1} & & c_{rk} \\ & r_1 & \\ & \vdots & \\ & \tilde{r}_p & \end{array}\right],$$

where (if we denote $\frac{k+r}{p} = t$),

$$\begin{aligned} r_1 &= (c_{(r+1)1}, c_{(r+1)2}, \ldots, c_{(r+1)t}, 0, \ldots, 0) \\ r_2 &= (0, \ldots, 0, c_{(r+1)(t+1)}, \ldots, c_{(r+1)2t}, 0, \ldots, 0) \\ &\vdots \\ r_p &= (0, \ldots, 0, c_{(r+1)(p(t-1))}, \ldots, c_k). \end{aligned}$$

Finally, we modify the last row by adding the exclusive-or of the global parity checks.

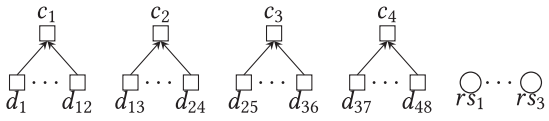
$$\tilde{r}_p = r_p + g_{k+1} + \cdots + g_{k+r}$$

One may note that the locality parameter for this code $\ell = \frac{k+r}{p}$, which is lower than that for Optimal Cauchy LRCs. In the event that $p \nmid (k+r)$, we may simply divide the $k+r$ data and global parities as evenly as possible among the p local checks.

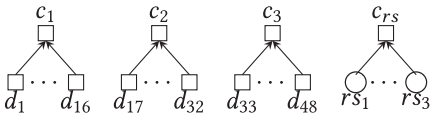
8 EXPERIMENTS AND ANALYSIS

We now use the following suite of practical measures of LRC quality to compare various codes:

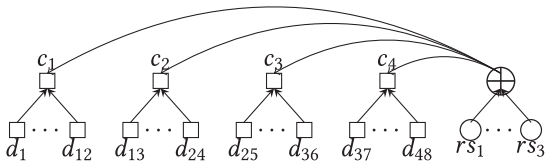
(1) Average degraded read cost (Definition 5.10).
(2) Average repair costs (Definitions 5.11).



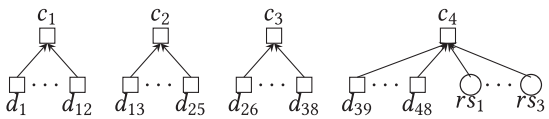
(a) 48-of-55 Azure-LRC



(b) 48-of-55 Azure-LRC + 1



(c) 48-of-55 Optimal Cauchy LRC



(d) 48-of-55 Uniform Cauchy LRC

Fig. 4. Different LRC constructions used in our evaluation.

(3) Reliability against random erasure patterns.
(4) Maximum tolerable failure rate for desired durability level (number of 9s).
(5) Comparison of reliability against the information theoretic limit (i.e., against MR-LRCs).
(6) A practical computation of **mean-time-to-data-loss (MTTDL)**.

We compare well-known LRC constructions including distance-optimal LRCs (our own Optimal Cauchy LRCs) and novel codes like our Uniform Cauchy LRCs. We compare these codes for the following parameter settings, all of which have $< 20\%$ storage overhead:

LRCs used in experiments. The most popular deployed LRCs we came across are the Xorbas-LRC [46], Azure-LRC [29], and Azure-LRC+1 [35] constructions. Xorbas-LRC has the advantage of being distance-optimal, but it is only shown to be so in very specific parameter settings and, thus, we cannot include them in our analysis of codes in more general parameter regimes. Other constructions of distance-optimal LRCs have been shown in works such as References [35] and [51], however, producing generator matrices for these codes is cumbersome, and they are not ideal candidates for a practical analysis. Instead, we use our novel and simple construction of distance-optimal codes, called Optimal Cauchy LRCs, as examples of distance-optimal codes in our evaluation. The Azure-LRC was one of the first LRCs proposed and is reportedly used at Microsoft Azure [29]. We illustrate these codes for the 48-of-55 parameter setting in Figure 4.

Azure-LRC divides the data blocks into equally spaced local groups, with each local group having a local parity. Each local parity can be the XOR of the data blocks in that local group. The global parities are not protected by any local parities. Figure 4(a) gives a representation of the Azure-LRC construction. Note that Azure-LRC is not a *true* LRC, since the global parities are not locally

Table 1. Wide LRC Schemes Used to Compare Different LRC Constructions

Scheme	n	k	r	p	Rate
24-of-28	28	24	2	2	$\frac{24}{28} = 0.857$
48-of-55	55	48	3	4	$\frac{48}{55} = 0.872$
72-of-80	80	72	4	4	$\frac{72}{80} = 0.9$
96-of-105	105	96	5	4	$\frac{96}{105} = 0.914$

Each scheme is chosen such that rate >= 0.85.

recoverable (i.e., $\ell = k$). In early works, locality for global parities was not considered a requirement for LRCs, but most recent works enforce it. However, we include this code in our analysis due to its popularity and because it is actually used in practice (and our analysis shows that it gives great performance!).

Azure-LRC+1 is an optimization to the Azure-LRC proposed in Reference [35]. Azure-LRC suffers from an expensive MDS-level reconstruction on the failure of any of the global parities. To prevent this, Azure-LRC+1 forms a local group of the global parities and protects them using a local parity. Figure 4(b) captures the Azure-LRC+1 construction. Note that the Azure-LRC+1 construction in Reference [35] introduces an additional local parity to protect the global parities without removing any of the existing parity blocks. This construction changes the storage overhead of Azure-LRC+1 in comparison to other constructions. We deliberately choose to keep the numbers k-of-n the same across all schemes, otherwise, we are comparing codes with different redundancy, which in our view is not a fair comparison. Thus, in our construction of Azure-LRC+1, we choose to remove one local parity protecting data blocks (as compared to removing a global parity, since removing a local parity has a less adverse effect on the reliability) and add a local parity protecting global parity blocks.

Figure 4(c) and Figure 4(d) are representations of the Optimal Cauchy and Uniform Cauchy constructions, which have been described comprehensively in previous sections.

Parameter regimes for comparison. We select four different widths representing wide LRCs, details of which are in Table 1. For an apples-to-apples comparison, we freeze the size of the data, k, the size of the code n, the number of global parities r, and the number of local parities, p. Since a (20, 17) MDS code is reportedly in use [4], we explore schemes where $24 \leq n \leq 105$ (deliberately starting from n close to 20, and codes with rate strictly higher than 0.85). The entire set of results is presented in Table 2, but we highlight the main points below.

Uniform Cauchy LRC has the smallest locality. We first compare the locality of the different LRC constructions. Recall (Definition 5.3) that locality refers to the maximum number of blocks to be read for reconstruction of a single block. Since Azure-LRC requires reading all data blocks to reconstruct any failed global parity, its locality is the highest. This is followed by Azure-LRC+1, whose local groups are larger than the other constructions (since we have constrained p). In fact, for 24-of-28, Azure-LRC+1 has the same locality as Azure-LRC due to having only 1 local group that spans all data blocks. Optimal Cauchy LRC evenly divides the data blocks, but requires each local group to contain *all* the global parities (we need this to prove distance optimality, as explained in Section 6.2), making its locality $\left\lceil \frac{k}{p} \right\rceil + g$. Uniform Cauchy LRC has the lowest locality $\left\lceil \frac{n}{p} \right\rceil$, since it uniformly divides n blocks into p groups.

Azure-LRC has the lowest average degraded read cost (*ADRC*), closely followed by Uniform Cauchy LRC. The *ADRC* is calculated only for the data blocks (Definition 5.10) and therefore is directly proportional to the size of local groups. Therefore, since Azure-LRC has the smallest local groups, it also has the smallest *ADRC*. Although, as the stripes become wider, the difference

Table 2. This Table Captures All the Analytical Metrics Used to Compare the Different LRC Constructions across the Different Wide LRC Parameters Described in Table 1

Locality	**Azure-LRC**	**Azure-LRC+1**	**Optimal Cauchy LRC**	**Uniform Cauchy LRC**
$n = 28, k = 24, r = 2, p = 2$	$\ell = 24$	$\ell = 24$	$\ell = 12 + 2 = 14$	$\ell =$**13**
$n = 55, k = 48, r = 3, p = 4$	$\ell = 48$	$\ell = 16$	$\ell = 12 + 3 = 15$	$\ell =$**13**
$n = 80, k = 72, r = 4, p = 4$	$\ell = 72$	$\ell = 24$	$\ell = 18 + 4 = 22$	$\ell =$**19**
$n = 105, k = 96, r = 5, p = 4$	$\ell = 96$	$\ell = 32$	$\ell = 24 + 5 = 29$	$\ell =$**26**
Avg. degraded read cost (*ADRC*)	**Azure-LRC**	**Azure-LRC+1**	**Optimal Cauchy LRC**	**Uniform Cauchy LRC**
$n = 28, k = 24, r = 2, p = 2$	$\frac{24\times12}{24} =$**12**	$\frac{24\times24}{24} = 24$	$\frac{12\times14}{12} = 14$	$\frac{26\times13}{26} = 13$
$n = 55, k = 48, r = 3, p = 4$	$\frac{48\times12}{48} =$**12**	$\frac{48\times16}{48} = 16$	$\frac{48\times15}{48} = 15$	$\frac{13\times12+35\times13}{48} = 12.72$
$n = 80, k = 72, r = 4, p = 4$	$\frac{72\times18}{72} =$**18**	$\frac{72\times24}{72} = 24$	$\frac{72\times22}{72} = 22$	$\frac{72\times19}{72} = 19$
$n = 105, k = 96, r = 5, p = 4$	$\frac{96\times24}{96} =$**24**	$\frac{96\times32}{96} = 32$	$\frac{96\times29}{96} = 29$	$\frac{78\times25+18\times26}{96} = 25.18$
Avg. repair cost 1 failure (ARC_1)	**Azure-LRC**	**Azure-LRC+1**	**Optimal Cauchy LRC**	**Uniform Cauchy LRC**
$n = 28, k = 24, r = 2, p = 2$	$\frac{26\times12+2\times24}{28} =$**12.85**	$\frac{25\times24+3\times2}{28} = 21.64$	$\frac{28\times13}{28} = 13$	$\frac{28\times13}{28} = 13$
$n = 55, k = 48, r = 3, p = 4$	$\frac{52\times12+3\times48}{55} = 13.96$	$\frac{51\times16+4\times3}{55} = 15.05$	$\frac{55\times15}{55} = 15$	$\frac{13\times12+42\times13}{55} =$**12.76**
$n = 80, k = 72, r = 4, p = 4$	$\frac{76\times18+4\times72}{80} = 20.7$	$\frac{75\times24+5\times4}{80} = 22.75$	$\frac{80\times22}{80} = 22$	$\frac{80\times19}{55} =$**19**
$n = 105, k = 96, r = 5, p = 4$	$\frac{100\times24+5\times96}{105} = 27.42$	$\frac{99\times32+6\times5}{105} = 30.45$	$\frac{105\times29}{105} = 29$	$\frac{81\times25+24\times26}{105} =$**25.22**
Avg. repair cost 2 failures (ARC_2)	**Azure-LRC**	**Azure-LRC+1**	**Optimal Cauchy LRC**	**Uniform Cauchy LRC**
$n = 28, k = 24, r = 2, p = 2$	30.66	43.46	32.12	**27.92**
$n = 55, k = 48, r = 3, p = 4$	35.49	39.22	36.93	**33.85**
$n = 80, k = 72, r = 4, p = 4$	52.80	59.38	54.82	**49.22**
$n = 105, k = 96, r = 5, p = 4$	70.68	79.73	74.50	**67.69**
Normalized MTTDL comparison	**Azure-LRC**	**Azure-LRC+1**	**Optimal Cauchy LRC**	**Uniform Cauchy LRC**
$n = 28, k = 24, r = 2, p = 2$	0.64×	0.14×	0.50×	**1.00**
$n = 55, k = 48, r = 3, p = 4$	0.99×	0.97×	**1.01**×	1.00
$n = 80, k = 72, r = 4, p = 4$	0.99×	0.97×	0.49×	**1.00**
$n = 105, k = 96, r = 5, p = 4$	0.99×	0.96×	0.96×	**1.00**

Details of this comparison are described in Section 8. The takeaways are that for all metrics except average degraded mode read cost (in which it is < 9% worse than the best LRC), Uniform Cauchy LRCs outperform other LRCs (including the Optimal Cauchy LRC). Another surprising result was that Azure-LRC outperforms Azure-LRC+1 despite its global parities having no local parities protecting them.

between the local group sizes of Azure-LRC and Uniform Cauchy LRC starts reducing. This reflects in the reduction of the ratio of the *ADRC* of Uniform Cauchy LRCs compared to Azure-LRCs. In particular, the *ADRC* of Uniform Cauchy LRC is about 8% more than Azure-LRC for 24-of-28, but is only about 5% more for 96-of-105. Azure-LRC+1 has the highest *ADRC* followed by Optimal Cauchy LRC, both owing to their larger local repair groups.

Average repair cost (ARC) is lowest for Uniform Cauchy LRC. Very similar to the *ADRC*, the ARC_1 and ARC_2 (Definition 5.11) are the degraded read cost for all n blocks of a stripe for 1 block failure and 2 block failures, respectively. We choose to showcase both ARC_1 and ARC_2, because the wider the LRC, the higher the likelihood that more than one failure can occur in a stripe, as explained in Section 3. Here, the disadvantage of Azure-LRC turns into the advantage for Uniform Cauchy LRC. Specifically, the degraded read cost for the global parities involves reading all data blocks in the case of Azure-LRC. For Uniform Cauchy LRC, the degraded read cost of global parities is the same as a data block, both of which are only slightly higher than the data block reconstruction cost of an Azure-LRC. The exception is 24-of-28, where Azure-LRC has a slightly lower ARC than Uniform Cauchy LRC, owing to these specific parameters, but this, too, becomes favorable for Uniform Cauchy LRCs in the case of ARC_2.[3] Across the schemes, as the number of data blocks increases, the ARC of Azure-LRC reduces, but at the same time, any additional global parity increases the *ARC* significantly. The difference between the Azure-LRC+1 and Optimal Cauchy LRC is lower than their difference in *ADRC*. This is due to Azure-LRC+1 having a very low-cost global parity reconstruction. Nevertheless, the Azure-LRC+1 and Optimal Cauchy LRCs have the highest and second-highest ARCs among the four constructions.

[3]We refrain from detailing the calculation of ARC_2, because it is significantly more complex than ARC_1.

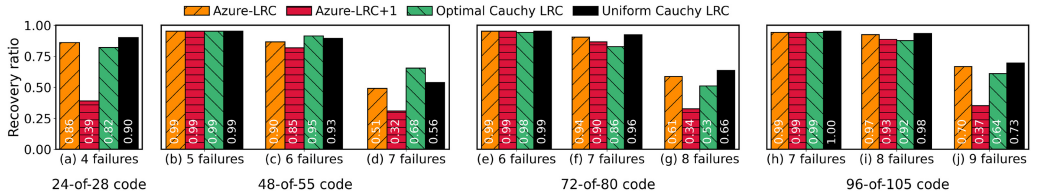


Fig. 5. This plot compares the durability of the different LRC constructions (Azure-LRC, Azure-LRC+1, Optimal Cauchy, and Uniform Cauchy) by measuring their ability to recover from random failures (only values where some recovery ratios were less than one are shown). We evaluate all wide LRCs discussed in evaluation ranging from 24-of-28 to 96-of-105. Except in 48-of-55, we see that Uniform Cauchy has the best durability. Optimal Cauchy has the best durability in 48-of-55. Surprisingly, Azure-LRC has better durability compared to Azure-LRC+1 even though global parities of Azure LRCs are not covered by a local parity.

Uniform Cauchy LRCs have the best random failure tolerance. We compare the LRC constructions empirically by evaluating their durability against random failures (including when the number of failures are $\geq d$ when d is the distance of the code). For each code, we choose various values of i and conduct a Monte Carlo experiment in which i blocks are removed uniformly at random (without replacement) from the n blocks of stripe. Then, data recovery is attempted. The more times recovery can succeed, the more durable the code. For each random failures experiment, at least 1 million recoveries from unique block failure combinations were attempted. The exception was 4 random failures for 24-of-28, where all combinations (20,475) were exhaustively checked.

Figure 5 shows the results of the various failure scenarios on each of the four schemes across all four LRCs. Uniform Cauchy LRC outperforms all other LRC constructions in each scenario, for each scheme, except 48-of-55. In 48-of-55, the Optimal Cauchy LRC has the highest recoverability ratio. The intuition behind Uniform Cauchy LRC's superior performance is that when a high number of failures happen uniformly at random, it is more likely that at least one failure occurs in each local repair group. This results in all local parities contributing to the reconstruction process, leading to a higher success rate.

Uniform Cauchy LRC has the highest failure rate tolerance for desired durability. In a practical setting, data durability is often specified in terms of a **service level objective (SLO)**, colloquially known as "number of 9s." For instance, four 9s of computed durability implies 99.99% durability, which in turn means that at most 1 in 10,000 objects (or stripes, or files, depending on the granularity of the definition) will be irrecoverably lost in a year. Large data clouds such as Google [42] and Amazon S3 [2] offer 11 9s of computed durability, which amounts to at most 1 object irrecoverably lost per 10^{11} objects in a year.

For various SLO obligations (specifically, 9, 10, and 11 9s), we experimentally determine the maximum *block failure probability* at which each code can still satisfy the SLO. In our analysis, block failures happen independently and uniformly at random (i.e., the number of failures follows a binomial distribution) with a fixed *block failure probability*. For each code and SLO pair, we conduct Monte Carlo simulations (of one million trials) with increasing block failure probabilities. This way, for each pair, we find the maximum block failure probability such that the code can still satisfy the SLO. Figure 6 shows our evaluation for the 48-of-55 LRCs using the various constructions with the desired SLOs set to 9 9s, 10 9s, and 11 9s. Uniform Cauchy LRCs have the highest failure tolerance for all desired SLOs, with Optimal Cauchy LRCs in a close second place. Azure-LRC+1 can tolerate the fewest number of simultaneous block failures before violating the SLOs, which corroborates our findings in Figure 5. Uniform Cauchy LRCs can tolerate 1.22× as many simultaneous block failures for 9 9s, 1.25× for 10 9s, and 1.24× for 11 9s compared to Azure-LRC+1.

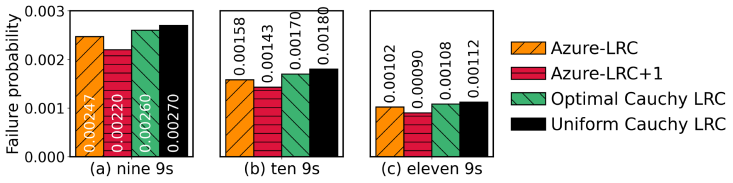


Fig. 6. This plot compares the maximum block failure probability (assuming all blocks can fail independently, uniformly at random) to provide desired durability level defined in number of 9s. Uniform Cauchy LRCs can tolerate the highest block failure probability, whereas Azure-LRC+1 has the lowest tolerance. Maximum block failure probabilities for Uniform Cauchy LRCs are between 1.03×–1.05× Optimal Cauchy LRCs, 1.09×–1.13× Azure-LRCs, and 1.22×–1.25× Azure-LRC+1 for different realistic durability SLOs.

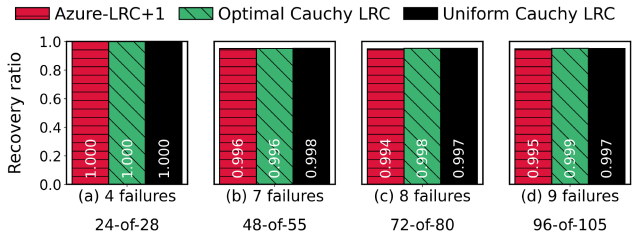


Fig. 7. **Comparing performance with MR-LRC.** In this plot, we show the results of a Monte Carlo experiment where p failures were forced (one in each local group) and another $n - k - p$ failures were distributed randomly across remaining blocks. MR-LRCs can recover from all such failure patterns. Azure-LRC+1, Optimal Cauchy, and Uniform Cauchy are all > 99% as durable as an MR-LRC in this scenario for our choice of coefficients. Azure-LRC is not shown, because it does not have a local group covering its global parities, making it unsuitable for this comparison.

Simple choices of coefficients give *almost* MR-LRCs in our parameter regime. We construct all our generator matrices using Cauchy matrices. We then conduct an experiment to compare our codes to a *hypothetical MR-LRC* of the exact same design. We say "hypothetical" because although there is proof that such an MR-LRC can exist [20, 24], currently there does not exist a deterministic way to construct such a code. Nevertheless, we can precisely characterize the erasure patterns that can be recovered by such MR-LRCs. Simply put, an MR-LRC with p local parities and r global parities can recover *any* pattern of $r + p$ failures, *as long as* there is at least one failure in each local repair group (so each local parity may contribute towards the reconstruction). An LRC that has this property is an MR-LRC.

We conduct an experiment in which we *plant* one failure in each local group (i.e., p total planted failures) and then add another r failures at random. Then, we attempt recovery. We find that at least with coefficients we chose (all derived from Cauchy matrices), *all* the codes we tested were very close to being MR-LRCs, as shown in Figure 7. So, even if it is hard to construct MR-LRCs, in practical parameter settings, it is not hard to realize many of their benefits using common code constructions.

We do point out that even though all code constructions were close to being MR-LRCs, this does not mean that they were all equally durable against random erasures. This is because the shape of the code affects the probability that $r + p$ random failures will spread out so each local repair group gets at least one failure. Indeed, this provides some intuition as to why Uniform Cauchy LRCs perform the best against random erasures. It is because the evenly sized local repair

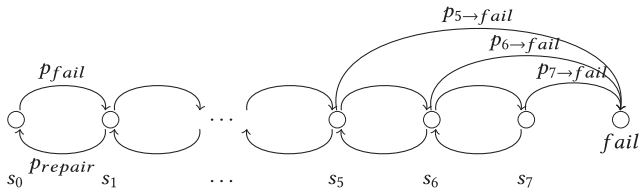


Fig. 8. MTTDL Markov Chain for a 48-of-55 code. The probabilities $p_{5\to fail}$, $p_{6\to fail}$, and $p_{7\to fail}$ are added from Figure 5 (b, c, d) where they are determined empirically. The probabilities $p_{i\to fail}$ for $i < 5$ are not pictured because they are zero.

groups maximize the probability that each local repair group will see at least one failure (recall MR-LRC Definiton 5.9).

Mean-time-to-data-loss (MTTDL). MTTDL (for a stripe) has been a canonical metric for reliability in the coding community [16]. It is modeled using a continuous time Markov chain in which each state represents the number of erasures in a stripe. There is a final absorbing state, denoted *fail*, which represents a data loss/unavailability event. The MTTDL for a coding scheme is simply the mean time until a Markov chain starting at state zero reaches the absorbing state, i.e., the expected time until a healthy stripe experiences a data unavailability event. Each state representing e failures, denoted s_e, may transition to the states s_{e-1}, s_{e+1} or *fail*. These transition probabilities are usually modelled using exponential distributions representing *repair time* (which captures the transition from s_e to s_{e-1}), *failure probability* (which captures the transition from s_e to s_{e+1}), and *complete failure probability* (which captures the transition from s_e to *fail*). These theoretical models are ubiquitous in modeling stripe reliability for MDS codes, but they have also been used to study the reliability of LRCs [29, 46, 48]. In our work, replace the *modelled* transition probabilities with *observed* ones to get a better estimate of the MTTDL of our codes.

We accomplish this by supplementing our theoretical model with empirical data in the following way: For each state s_e, we add a transition to the irrecoverable data loss state with probability $p_{e\to fail}$, where $p_{e\to fail}$ is probability that a stripe with e randomly distributed failures cannot recover the data, as shown in Figure 8. To be clear, to model MTTDL this way, we need the explicit generator matrix of our code and then to conduct the random failures experiment to generate data. While no model is perfect, we believe that this is a more realistic evaluation of MTTDL of an LRC, since we meaningfully account for the observed durability of the explicit code (i.e., accounting for coefficients).

Table 2 shows the MTTDL comparisons for different LRC constructions. Since we are interested in the relative comparison of MTTDLs, we show the ratio of each LRC construction with the Uniform Cauchy LRC MTTDL (where MTTDL is calculated as a function of p_{fail} and p_{repair} whose values can differ significantly based on specific cluster size, architecture, disk makes/models, etc.). We assume that nodes fail uniformly, and although we did not explicitly measure this in our storage cluster, the risk-diversity policies are designed to ensure that disks on which data is striped fail independently. For proprietary reasons, we cannot reveal the exact values of p_{fail} and p_{repair}. The trends between different LRC constructions hold for p_{fail} values derived from a failure rate of 16% as shown in References [32–34] and p_{repair} on the order of a few hours. Although p_{repair} can differ based on the architecture of the code, those differences are negligible in comparison to the wait-time for the detection of the missing blocks in a stripe, as explained in Section 2. We find that Uniform Cauchy LRC achieves the highest MTTDL (up to a factor of 100× in our evaluation), while Azure-LRC+1 has the lowest MTTDL values. This is in line with our observations that Uniform Cauchy LRCs and Azure-LRC+1 have, respectively, the highest and lowest performance in

(a) Maintenance-robust-efficient deployment with $z = 3$. (b) Maintenance-robust deployment with $z = 2$.

Fig. 9. Different deployment patterns for the (9 ,4, 2)–LRC from Figure 3.

reconstructing random erasures, and also the lowest and highest average locality. This is unsurprising, since MTTDL of an LRC is inversely proportional to its average locality [48].

9 MAINTENANCE-ROBUST DEPLOYMENT

When deploying an erasure code in a large-scale storage cluster, there are several placement constraints that need to be met to ensure adequate data reliability. For example, usually no two blocks of the same stripe are put on the same disk, server, rack, or at times even racks powered by the same power source to improve data reliability. Placement restrictions have been studied recently in Reference [32], and in this work, we comment that placement constraints can be considered during code design as well.

In a storage cluster, the smallest unit in which maintenance such as kernel/firmware/hardware upgrades can be performed is known as a *maintenance zone*. The design of maintenance zones affects reliability, because all servers/devices in a single maintenance zone can be turned off simultaneously during a maintenance event. So, we would like to have each stripe intersect a maintenance zone as little as possible, making it robust to the maintenance events in a real storage cluster. In an ideal world, we could make maintenance zones as small as possible (e.g., a single rack) to limit the impact of maintenance events on reliability. But this is not practically feasible, as it would increase the operational toil of maintenance tasks. One reason for this is: After every maintenance task (belonging to a single maintenance zone) cluster-wide (and therefore time-consuming and expensive) data reachability and reliability tests are usually conducted to ensure that the maintenance activity was completed without unforeseen events. If maintenance zones are too small (say, an individual server), then a large-scale cluster of tens-of-thousands of servers would be in perennial testing activity. This severely limits the number of maintenance zones, and in practice, storage administrators have informed us that the total number of maintenance zones in a single cluster are typically restricted to below 20. This means that a maintenance event can affect a large number of stripes, and hence it is important to deploy stripes in a manner that is robust to maintenance events.

One desirable property of a code deployment might be that no maintenance event that affects a single zone should render any stripe unrecoverable. We name this *maintenance-robust deployment*. Since maintenance events are foreseeable, and usually performed one zone at a time, a maintenance-robust deployment would ensure high availability.

In the context of LRCs, we can further optimize deployment strategies to increase the likelihood of cost-efficient local repairs. For example, if we guarantee that every maintenance zone has at most one block from each local group, then all degraded reads during a maintenance event would only require local repairs. We term such a deployment *maintenance-robust-efficient deployment*. Ideally, we want all LRC deployments to be maintenance-robust-efficient deployments.

To clarify these definitions, consider the $(9, 4, 2)$–LRC from Figure 3. If the number of maintenance zones, say, z, is 3, then we can assign each block of data from every local repair group to a different maintenance zone (i.e., a maintenance-robust-efficient deployment), as shown in Figure 9(a). Here, during *any* maintenance event, the degraded read cost for the data blocks stored

in that maintenance zone is still $\ell = 2$. However, if $z = 2$, as shown in Figure 9(b), then we cannot guarantee local repairs in all cases. As a best case, we could have $z_1 = d_1, d_3, rs_1, c_3$ and put the rest of the data in z_2. Suppose z_1 undergoes maintenance, the data blocks d_1 and d_3 can be repaired locally using blocks d_2, c_1 and d_4, c_2, respectively. After the data is repaired, rs_1 can be repaired globally using the data blocks and rs_2 can be repaired locally using c_3 and rs_1. Suppose z_2 undergoes maintenance, the data can be repaired by performing global repairs using d_1, d_3, rs_1, and rs_2. Finally, the local parity c_3 can be repaired using rs_1 and rs_2. Thus, with $z = 2$, a $(9, 4, 2)-$LRC can only be deployed in a maintenance-robust manner, where global repairs are required for reconstructing data. In general, when $z \geq \ell+1$ for an LRC, it can guarantee maintenance-robust-efficient deployment.

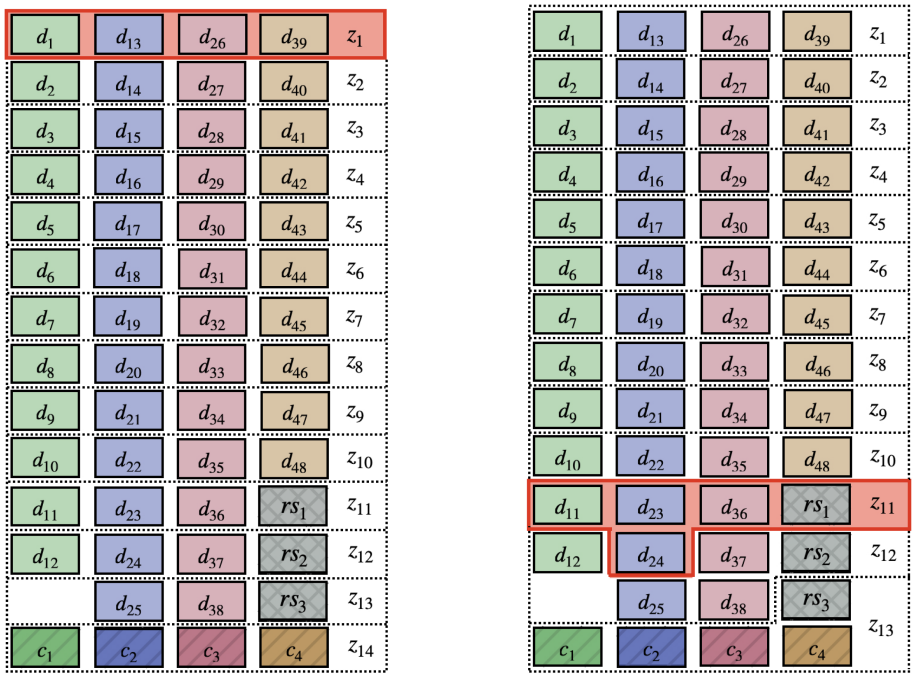
We remark that the concept of maintenance zones is particularly important in practical settings, since wider codes (i.e., codes with less storage overhead) require more maintenance zones to just ensure maintenance-robust deployment. This is one of the limitations that make it challenging to deploy wide codes. In fact, even when maintenance-robust deployment is possible, code design and the exact number of zones can have a meaningful impact on the average repair cost during a maintenance event. In the next section, we use the 48-of-55 code configuration with a moderate number of zones (between 8 and 17), to quantify this effect.

Average Maintenance Cost (AMC). Figure 10(a) shows the maintenance-robust-efficient deployment of the 48-of-55 Uniform-Cauchy LRC using 14 maintenance zones. Each z_i has exactly one block from each local group with the exception of z_{13}, which has no blocks from the first local group because of the unevenly sized groups. We now consider the average read cost for reconstructing data in a maintenance zone. Note that this cost depends not only on the code, but on the number of maintenance zones and the specific choice of blocks in the maintenance zone. In the deployment shown in Figure 10(a), the unavailability of every maintenance zone except z_{13} requires recovering four blocks. As a concrete example, consider the case when z_1 is unavailable: Repairing d_1 requires reading 12 blocks, whereas repairing d_{13}, d_{26}, and d_{39} requires reading 13 blocks each, resulting in an average read cost of 12.75.

Now consider a deployment of the same LRC using 13 zones as shown in Figure 10(b). With 13 zones, it is clear that a maintenance-robust-efficient deployment is not possible, as at least one zone would need to contain more than four blocks (i.e., it would need to contain two blocks from the same local group). We chose a deployment that has five blocks in three zones, z_{11}, z_{12}, z_{13}, as depicted in Figure 10(b). Suppose z_{11} is unavailable, repairing d_{23} and d_{24} from the same local group would require one global repair followed by one local repair. Thus, even with just one lesser maintenance zone overall, this maintenance-robust deployment has an average read cost of 14.67.

Continuing this way, we may quantify the *overall* impact of the number of maintenance zones on repair performance. We calculate the read cost for *all* maintenance zones in each deployment of a code and compute a weighted average, which we call the ***AMC* (average maintenance cost)**. For the sake of this average, we consider the probability of each maintenance zone being unavailable as *proportional* to its size. For concreteness, note that this quantity is the regular ARC_1 when the number of maintenance zones is n, and each block is in its own maintenance zone (a careful reader may observe that it is in fact equal to ARC_1 when the number of maintenance zones is $z \geq \ell + 1$). We analytically evaluate this cost, and the results are shown in Figure 11.

Figure 11 shows the average *AMC* for all the LRC constructions we have evaluated in this article for the 48-of-55 configuration. The x-axis denotes the number of maintenance zones in a deployment, whereas the y-axis denotes the *AMC*. We vary z between 8 and 17 for each scheme. We observe that Azure-LRC performs well here and has the lowest *AMC* until the number of zones is ≥ 14, at which point the Uniform Cauchy LRC has the lowest *AMC*. This is expected, because Azure-LRCs has the smallest local group size for the data blocks, followed by Uniform Cauchy



(a) Maintenance-robust-efficient deployment with 14 zones. (b) Maintenance-robust deployment with 13 zones.

Fig. 10. **Uniform Cauchy 48-of-55 deployment with $z = 13$ and $z = 14$ zones.** Figures 10(a) shows maintenance-robust-efficient deployment of 48-of-55 LRC with $z = 14$. Whenever any zone is unavailable (such as z_1 shown above), all blocks can be repaired locally. With 13 zones as shown in Figure 10(b), the deployment is maintenance-robust, but not maintenance-robust-efficient. z_{11}, z_{12}, z_{13} each have one extra block from local groups 2, 3, and 4, respectively. When z_{11} is unavailable, a global repair needs to be issued in the second local group, because it has two unavailable blocks (d_{23}, d_{24}), but only one local parity.

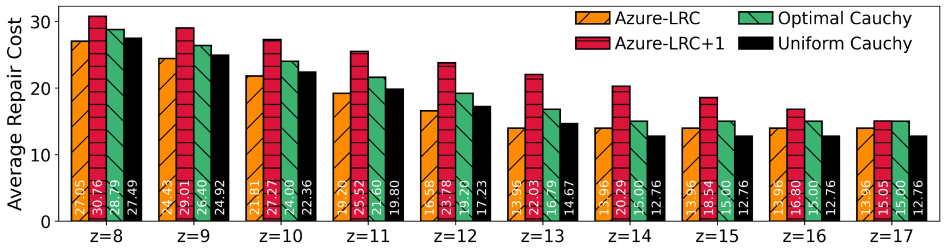


Fig. 11. This plot shows the average repair cost (*ARC*) for the deployment of an LRC across different number of maintenance zones. Except for $z = 14$, Azure-LRCs has the lowest maintenance *ARC*. Note that for an LRC, when $z \geq \ell + 1$, the maintenance $ARC = ARC_1$ from Table 2. Overall, the lowest maintenance $ARC = 12.76$ for the Uniform Cauchy LRC deployed across 14 or more maintenance zones.

LRC. But, Azure-LRC *AMC* is at most 5% better than Uniform Cauchy LRC when $z \leq 13$, whereas the Uniform Cauchy LRC *AMC* is ≈10% better than Azure-LRC *AMC* for $z \geq 14$. Azure-LRCs will always have a "flip" point when increasing the number of maintenance zones compared to Uniform Cauchy LRCs. In the example, we have shown, beyond $z = 13$, Azure LRCs starts making many global repairs, disallowing the ARC to go down further. For Uniform Cauchy LRCs, since

the global parities are also a part of a local group, their repair happens locally, allowing Uniform Cauchy LRCs to have a lower ARC when $z = 14$. Both Optimal Cauchy and Azure-LRC+1 have designs that are less convenient for maintenance-robust-efficient deployment and suffer from high average *AMC* irrespective of the number of maintenance zones. Overall, with a moderate number of maintenance zones Uniform-Cauchy LRCs provide the lowest average *ARC* for wide stripes. More importantly, one may note that the number (and specific design) of maintenance zones can have a meaningful impact on degraded read performance of a code.

10 RELATED WORK

Local codes in practice. LRCs have been widely studied and deployed in the past decade due to their practical improvements over MDS codes. While Microsoft Azure and Facebook first published the commercial use of LRCs [29, 46], much academic research has also been conducted to find LRC constructions with desirable properties [1, 7, 12, 19, 43, 43, 48, 50, 52]. One area of research has been explicit constructions of distance-optimal codes (Definition 5.4) [7, 26, 30, 37, 39, 40, 51, 53, 55]. Tamo and Barg [51] were among the first to provide a general construction of distance-optimal LRCs (with small field sizes), but with some constraints on the allowable parameters. This construction was further generalized by Kolosov et al. [35] for a wide range of parameters. Recent years have seen a surge in research on MR-LRCs [5, 6, 13, 17, 18, 20–24, 38]. Recently, Gopi et al. [20] provided an explicit construction of MR-LRCs that is broad, but does not cover our parameter regime.

Wide codes in practice. Wide codes are known to be deployed in two commercial settings, VAST [54] and Backblaze [4], and recent works have studied wide codes in academia. Haddock et al. analyze wide codes' performance using GPUs [25]. Li et al. show local erasures codes in the context of hard-disk drives [36]. More recently, Hu et al. [27] study the performance issues in repair, encoding, and update performance of wide codes. Our practically inspired work adds to this literature by studying the tradeoffs of wide LRCs from construction to deployment.

Constructions of distance-optimal LRCs. Constructing distance-optimal LRCs is a well-studied problem. At first, only constructions whose alphabet size was exponential in the block-length were known [49, 53]. These do not yield codes that may be used in practical settings where the alphabet size q is fixed at 256. The most well-known construction of distance-optimal LRCs may be found in References [35, 51]. To get truly optimal codes, the codes of Reference [51] require that $\ell+1|n$, and none of the codes we use in our experiments meet this restriction. Distance-optimal LRCs with very general parameters were finally shown in Reference [35]. Our construction of optimal Cauchy LRCs adds to this literature, because it is remarkably simple and yields many codes in practical parameter regimes.

Data placement constraints in systems. The original Azure-LRC paper [29] discussed placement of blocks across failure domains and upgrade domains but focused on just achieving maintenance-robustness, not necessarily efficiency. Copysets [14] and Flat Datacenter Storage [41] discuss maintenance-robust deployments of replicated data in a storage cluster. Our maintenance-robust deployment is a general framework that can be used for deployment of both replicated and erasure coded data. Recent work on disk-adaptive redundancy [32] has highlighted that data placement, especially in the context of erasure codes, is a highly constrained problem. Requiring fewer maintenance zones for a maintenance-robust-efficient deployment and lower ARC when a maintenance zone is unavailable are practically important considerations.

11 CONCLUSION

In this work, we show that many subtle factors can affect LRC reliability in real-world scenarios: coefficients in the generator matrix, design of local repair groups, and maintenance-robust-deployment. We show the value of an experiment-driven understanding of reliability, as it provides

us novel insights into design choices that have a meaningful impact on reliability. Indeed, this culminates in our construction of Uniform Cauchy LRCs, which outperform popular (and provably distance-optimal) codes in practice.

ACKNOWLEDGMENTS

We thank Patrick P. C. Lee and the anonymous reviewers at USENIX FAST for their valuable feedback and suggestions on an earlier version of this manuscript. We extend special thanks to Eric Schrock, Larry Greenfield, and numerous other researchers and engineers at Google.

REFERENCES

[1] Abhishek Agarwal, Alexander Barg, Sihuang Hu, Arya Mazumdar, and Itzhak Tamo. 2018. Combinatorial alphabet-dependent bounds for locally recoverable codes. *IEEE Trans. Inf. Theor.* (2018).
[2] Amazon. 2023. Amazon S3 FAQs. Retrieved from https://aws.amazon.com/s3/faqs
[3] Backblaze. 2013–2018. Disk Reliability Dataset. Retrieved from https://www.backblaze.com/b2/hard-drive-test-data.html
[4] Backblaze. 2013–2018. Erasure coding used by Backblaze. Retrieved from https://www.backblaze.com/blog/reed-solomon/
[5] S. B. Balaji and P. Vijay Kumar. 2015. On partial maximally-recoverable and maximally-recoverable codes. In *IEEE International Symposium on Information Theory (ISIT'15)*.
[6] Alexander Barg, Zitan Chen, and Itzhak Tamo. 2022. A construction of maximally recoverable codes. *Des., Codes Cryptog.* (2022).
[7] Alexander Barg, Itzhak Tamo, and Serge Vlăduţ. 2017. Locally recoverable codes on algebraic curves. *IEEE Trans. Inf. Theor.* (2017).
[8] Shimrit Ben-Yair. 2020. Updating Google Photos' storage policy to build for the future. Retrieved from https://blog.google/products/photos/storage-changes/
[9] Johannes Bloemer, Malik Kalfane, Richard Karp, Marek Karpinski, Michael Luby, and David Zuckerman. 1995. *An XOR-Based Erasure-Resilient Coding Scheme.* University of California, Berkeley.
[10] Eric Brewer. 2018. Spinning Disks and Their Cloudy Future. Retrieved from https://www.usenix.org/node/194391
[11] Eric Brewer, Lawrence Ying, Lawrence Greenfield, Robert Cypher, and Theodore T'so. 2016. *Disks for Data Centers.* Technical Report. Google.
[12] Viveck Cadambe and Arya Mazumdar. 2013. An upper bound on the size of locally recoverable codes. In *International Symposium on Network Coding (NetCod'13)*. IEEE.
[13] Han Cai, Ying Miao, Moshe Schwartz, and Xiaohu Tang. 2021. A construction of maximally recoverable codes with order-optimal field size. *IEEE Trans. Inf. Theor.* (2021).
[14] Asaf Cidon, Stephen Rumble, Ryan Stutsman, Sachin Katti, John Ousterhout, and Mendel Rosenblum. 2013. Copysets: Reducing the frequency of data loss in cloud storage. In *USENIX Annual Technical Conference (USENIX ATC 13)*. 37–48.
[15] Daniel Ford, François Labelle, Florentina I. Popovici, Murray Stokely, Van-Anh Truong, Luiz Barroso, Carrie Grimes, and Sean Quinlan. 2010. Availability in globally distributed storage systems.. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI'10)*.
[16] Garth Alan Gibson. 1991. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage.* Ph. D. Dissertation. University of California, Berkeley.
[17] Parikshit Gopalan, Guangda Hu, Swastik Kopparty, Shubhangi Saraf, Carol Wang, and Sergey Yekhanin. 2017. Maximally recoverable codes for grid-like topologies. In *28th Annual ACM-SIAM Symposium on Discrete Algorithms*.
[18] Parikshit Gopalan, Cheng Huang, Bob Jenkins, and Sergey Yekhanin. 2014. Explicit maximally recoverable codes with locality. *IEEE Trans. Inf. Theor.* (2014).
[19] Parikshit Gopalan, Cheng Huang, Huseyin Simitci, and Sergey Yekhanin. 2012. On the locality of codeword symbols. *IEEE Trans. Inf. Theor.* (2012).
[20] Sivakanth Gopi and Venkatesan Guruswami. 2022. Improved maximally recoverable LRCs using skew polynomials. *IEEE Trans. Inf. Theor.* (2022).
[21] Sivakanth Gopi, Venkatesan Guruswami, and Sergey Yekhanin. 2017. On maximally recoverable local reconstruction codes. In *Electronic Colloquium on Computational Complexity (ECCC'17)*.
[22] Sivakanth Gopi, Venkatesan Guruswami, and Sergey Yekhanin. 2020. Maximally recoverable LRCs: A field size lower bound and constructions for few heavy parities. *IEEE Trans. Inf. Theor.* (2020).
[23] Matthias Grezet, Thomas Westerbäck, Ragnar Freij-Hollanti, and Camilla Hollanti. 2019. Uniform minors in maximally recoverable codes. *IEEE Commun. Lett.* (2019).

[24] Venkatesan Guruswami, Satyanarayana V. Lokam, and Sai Vikneshwar Mani Jayaraman. 2020. -MSR codes: Contacting fewer code blocks for exact repair. *IEEE Trans. Inf. Theor.* (2020).
[25] Walker Haddock, Purushotham V. Bangalore, Matthew L. Curry, and Anthony Skjellum. 2019. High performance erasure coding for very large stripe sizes. In *Spring Simulation Conference (SpringSim'19)*.
[26] Kathryn Haymaker, Beth Malmskog, and Gretchen Matthews. 2016. Locally recoverable codes with availability $t <= 2$ from fiber products of curves. *arXiv preprint arXiv:1612.03841* (2016).
[27] Yuchong Hu, Liangfeng Cheng, Qiaori Yao, Patrick P. C. Lee, Weichun Wang, and Wei Chen. 2021. Exploiting combined locality for wide-stripe erasure coding in distributed storage. In *USENIX File and Storage Technologies Conference (FAST'21)*.
[28] Cheng Huang, Minghua Chen, and Jin Li. 2013. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *ACM Trans. Stor.* (2013).
[29] Cheng Huang, Huseyin Simitci, Yikang Xu, Aaron Ogus, Brad Calder, Parikshit Gopalan, Jin Li, and Sergey Yekhanin. 2012. Erasure coding in windows azure storage. In *USENIX Annual Technical Conference (ATC'12)*.
[30] Lingfei Jin. 2019. Explicit construction of optimal locally recoverable codes of distance 5 and 6 via binary constant weight codes. *IEEE Trans. Inf. Theor.* (2019).
[31] Saurabh Kadekodi. 2020. *DISK-ADAPTIVE REDUNDANCY: Tailoring Data Redundancy to Disk-reliability Heterogeneity in Cluster Storage Systems*. Ph. D. Dissertation. Carnegie Mellon University.
[32] Saurabh Kadekodi, Francisco Maturana, Sanjith Athlur, Arif Merchant, K. V. Rashmi, and Gregory R. Ganger. 2022. Tiger: Disk-adaptive redundancy without placement restrictions. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI'22)*.
[33] Saurabh Kadekodi, Francisco Maturana, Suhas Jayaram Subramanya, Juncheng Yang, K. V. Rashmi, and Gregory R. Ganger. 2020. PACEMAKER: Avoiding HeART attacks in storage clusters with disk-adaptive redundancy. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*.
[34] Saurabh Kadekodi, K. V. Rashmi, and Gregory R. Ganger. 2019. Cluster storage systems gotta have HeART: Improving storage efficiency by exploiting disk-reliability heterogeneity. In *USENIX File and Storage Technologies Conference (FAST'19)*.
[35] Oleg Kolosov, Gala Yadgar, Matan Liram, Itzhak Tamo, and Alexander Barg. 2020. On fault tolerance, locality, and optimality in locally repairable codes. *ACM Trans. Stor. (TOS)* (2020).
[36] Yin Li, Hao Wang, Xuebin Zhang, Ning Zheng, Shafa Dahandeh, and Tong Zhang. 2017. Facilitating magnetic recording technology scaling for data center hard disk drives through filesystem-level transparent local erasure coding. In *USENIX File and Storage Technologies Conference in 2017*.
[37] Jian Liu, Sihem Mesnager, and Lusheng Chen. 2017. New constructions of optimal locally recoverable codes via good polynomials. *IEEE Trans. Inf. Theor.* (2017).
[38] Shu Liu and Chaoping Xing. 2021. Maximally recoverable local reconstruction codes from subspace direct sum systems. *arXiv preprint arXiv:2111.03244* (2021).
[39] Gaojun Luo and Xiwang Cao. 2021. Constructions of optimal binary locally recoverable codes via a general construction of linear codes. *IEEE Trans. Commun.* (2021).
[40] Giacomo Micheli. 2019. Constructions of locally recoverable codes which are optimal. *IEEE Trans. Inf. Theor.* (2019).
[41] Edmund B. Nightingale, Jeremy Elson, Jinliang Fan, Owen Hofmann, Jon Howell, and Yutaka Suzue. 2012. Flat datacenter storage. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI'12)*. 1–15.
[42] Geoffrey Noer and David Petrie Moulton. 2021. How Cloud Storage Delivers 11 Nines of Durability—and How You Can Help. Retrieved from https://cloud.google.com/blog/products/storage-data-transfer/understanding-cloud-storage-11-9s-durability-target
[43] D. S. Papailiopoulos and A. G. Dimakis. 2012. Locally repairable codes. In *IEEE International Symposium on Information Theory (ISIT'12)*.
[44] K. V. Rashmi, Nihar B. Shah, Dikang Gu, Hairong Kuang, Dhruba Borthakur, and Kannan Ramchandran. 2013. A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster. In *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'13)*.
[45] David Reinsel, John Gantz, and John Rydning. 2018. The digitization of the world from edge to core. *Framing.: Int. Data Corp.* 16 (2018).
[46] Maheswaran Sathiamoorthy, Megasthenis Asteris, Dimitris Papailiopoulos, Alexandros G. Dimakis, Ramkumar Vadali, Scott Chen, and Dhruba Borthakur. 2013. XORing elephants: Novel erasure codes for big data. In *International Conference on Very Large Data Bases (VLDB'13)*.
[47] Seagate. 2018. The Digitization of the World from Edge to Core. Retrieved from https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf
[48] Mostafa Shahabinejad. 2018. Locally repairable linear block codes for distributed storage systems. Department of Electrical and Computer Engineering at the University of Alberta.

[49] Natalia Silberstein, Ankit Singh Rawat, O. Ozan Koyluoglu, and Sriram Vishwanath. 2013. Optimal locally repairable codes via rank-metric codes. In *IEEE International Symposium on Information Theory (ISIT'13)*.
[50] Itzhak Tamo and Alexander Barg. 2014. Bounds on locally recoverable codes with multiple recovering sets. In *IEEE Trans. Inf. Theor.* (2014)
[51] Itzhak Tamo and Alexander Barg. 2014. A family of optimal locally recoverable codes. *IEEE Trans. Inf. Theor.* (2014).
[52] Itzhak Tamo, Alexander Barg, and Alexey Frolov. 2016. Bounds on the parameters of locally recoverable codes. *IEEE Trans. Inf. Theor.* (2016).
[53] Itzhak Tamo, Dimitris S. Papailiopoulos, and Alexandros G. Dimakis. 2016. Optimal locally repairable codes and connections to matroid theory. *IEEE Trans. Inf. Theor.* (2016).
[54] VAST. 2019. Providing Resilience, Efficiently. Retrieved from https://www.usenix.org/node/194391
[55] Guanghui Zhang. 2020. A new construction of optimal (r, δ) locally recoverable codes. *IEEE Commun. Lett.* (2020).

Received 13 May 2023; revised 1 August 2023; accepted 12 September 2023