Alpha Entanglement Codes: Practical Erasure Codes to Archive Data in Unreliable Environments

Vero Estrada-Galiñanes*, Ethan Miller†, Pascal Felber* and Jehan-François Pâris‡
*University of Neuchâtel, 2000 Neuchâtel, Switzerland
†University of California, Santa Cruz, CA 95064, USA
‡University of Houston, Houston, TX 77204-3010, USA

***Abstract*—Data centres that use consumer-grade disks drives and distributed peer-to-peer systems are unreliable environments to archive data without enough redundancy. Most redundancy schemes are not completely effective for providing high availability, durability and integrity in the long-term. We propose alpha entanglement codes, a mechanism that creates a virtual layer of highly interconnected storage devices to propagate redundant information across a large scale storage system. Our motivation is to design flexible and practical erasure codes with high fault-tolerance to improve data durability and availability even in catastrophic scenarios. By "flexible and practical", we mean code settings that can be adapted to future requirements and practical implementations with reasonable trade-offs between security, resource usage and performance. The codes have three parameters. Alpha increases storage overhead linearly but increases the possible paths to recover data exponentially. Two other parameters increase fault-tolerance even further without the need of additional storage. As a result, an entangled storage system can provide high availability, durability and offer additional integrity: it is more difficult to modify data undetectably. We evaluate how several redundancy schemes perform in unreliable environments and show that alpha entanglement codes are flexible and practical codes. Remarkably, they excel at code locality, hence, they reduce repair costs and become less dependent on storage locations with poor availability. Our solution outperforms Reed-Solomon codes in many disaster recovery scenarios.**

I. Introduction

Redundancy schemes play a key role in the security and performance of storage systems. Many cloud-based centralised solutions use replication or RAID-like techniques [1], the latter often built with erasure codes [2] like Reed-Solomon (RS) codes. In decentralised systems, replication is widely used due to the higher bandwidth requirements of erasure coding, particularly to rebuild single-failures. Choosing a redundancy scheme, for the most part, involves finding a good compromise between reliability, performance, and costs of additional storage, repair and maintenance of redundancy. The estimation of these variables is a subtle task, especially for archival systems that are expected to maintain data accessibility and integrity for the long term in unreliable environments.

Some of the most remarkable works in the last 20 years of research propose scalable redundancy mechanisms to address availability and/or correlated failures: The Eternity Service (1996) [3], PASIS (2000) [4], FARSITE (2002) [5], Glacier (2005) [6] and Carbonite (2006) [7] and recently RESAR (2016) [8]. In the same period of time, the storage scale has been changing abruptly and such change suggest us to rethink the schemes used to store data redundantly.

The industry sector puts strong emphasis on storage costs, which means that their service usually rely on hybrid redundancy schemes. For instance, Google uses multi-cell replication to survive data centre failures, and RS encoding for local failures [9]. Facebook uses geo-replicated XOR-based codes to provide fault-tolerance at the data centre level, and RS encoding within a single data centre [10]. RS codes are optimal storage codes, however, the overall storage overhead of the redundancy scheme is significant when used in combination with replication.

RS codes are designed to protect data against massive correlated failures, but, in practice, the parameter values are restricted to a small range due to poor performance. A system that uses *RS(k,m)* codes tolerates m failures, but requires k I/O accesses and $k \cdot B$ bandwidth to repair a single failure of B bytes. Some solutions to reduce the repair overhead were proposed [11]–[14]. Real-world systems advocate small values: RS(6,3) in Google's single-cell scheme [9], RS(10,4) in Facebook's f4 system [10] and $k + m \leqslant 20$ in Microsoft Azure [13]. With settings like those mentioned, systems can keep data safe only with the existence of continuous monitoring and repairing operations that increase storage cost. Another limitation is that reliability requirements may change over time but RS parameters are not dynamic: data must be encoded again to change k and m.

In this work, we design a robust and flexible redundancy scheme that uses storage and bandwidth resources efficiently, accommodate future reliability needs and its implementation does not require "magic numbers". We present alpha entanglement codes, AE(α, s, p), a family of erasure codes that is fundamentally centered around the new concept of redundancy propagation. Thus, an entangled storage system would provide permanent storage with high availability, durability and integrity despite external and internal threats. This work generalizes our earlier entanglement algorithms [15], [16], including them as special cases, defines redundancy propagation, discusses centralised and decentralised entangled storage systems, and improves evaluations done in other works [17].

AE codes create redundancy by tangling (mixing) new data blocks with old ones, building entangled data chains that are woven into a growing mesh of interdependent content. The robust connectivity provides high fault-tolerance. The only assumption is that data are stored permanently, deletions are only possible at the beginning of the mesh. The propagation is controlled by the code parameters: (i) α determines the local connectivity, (ii) s and p determine the global connectivity of data blocks in the grid. Single-entanglements ($\alpha = 1$) compute 1 parity per data block, double-entanglements ($\alpha = 2$) compute 2 parities, triple-entanglements ($\alpha = 3$) compute 3 parities and so on. Although the storage overhead increases linearly with the number of parities per data block, the number of possible *data recovery paths* grows exponentially. Due to the redundancy propagation, AE codes reveal a combination of features previously unavailable for storage systems.

Alpha entanglements are irregular codes. In simple words, the fault tolerance of irregular codes [18] goes beyond the m failures tolerated in (k, m)-codes but the extra tolerated failures are not arbitrary. The usual metrics to measure irregular fault tolerance are based in the distribution of the failure patterns that the code cannot tolerate. We use a variation of previously proposed metrics [18], [19] to show the benefits of redundancy propagation. Larger values in α, s, and/or p increase the size of minimal erasure patterns, $|ME(x)|$, that cause the irrecoverable loss of x data blocks. For example, we will show that a pattern that cause the loss of two data blocks for code setting AE(3,1,4) is $|ME(2)| = 8$ and it becomes considerable larger, $|ME(2)| = 14$, for code setting AE(3,4,4). A remarkable aspect of redundancy propagation is that tuning the parameters (s,p) does not modify the storage overhead and that none of the three parameters can change the cost of a single failure, which is always repaired by XORing two blocks. Another characteristic is that alpha entanglements permit changes in the parameters without the need to encode the content again. This property opens the possibility of a dynamic fault-tolerance, which is an interesting feature for long-term storage systems. As far as we know, these characteristics are not found in state-of-the-art codes.

The next section provides the background and the motivation for this work. We present alpha entanglement codes in Section III and provide a guideline of how they can be adapted to diverse applications and storage architectures in Section IV. The results of our evaluation are presented in Section V. First, we investigate the impact of different code settings on fault tolerance and write performance. Second, we run simulations over millions of synthetically generated blocks to evaluate data loss, redundancy degradation and repair performance in catastrophic failures to compare alpha entanglement codes and (k, m)-codes. Finally, we show that alpha entanglement codes outperform Reed-Solomon codes when are used in unreliable environments with poor maintenance. Alpha entanglement codes excel at code locality, hence they reduce repair cost and are less dependent on storage locations with poor availability.

II. Background and Motivation

Mixing data is being actively investigated in network coding [20]–[22] as well as in storage applications [23]–[26]. More specifically, document entanglement [25], [26] was proposed for censorship resistant systems but as pointed by others an efficient and practical method is not straightforward to devise [23]. We revisited the problem in the context of fault-tolerant systems and proposed helical entanglement codes (HEC) [15]. The code computes the exclusive-or (XOR) of two blocks: a parity and a data block. Every new data block is entangled with three parities. The three outputs contribute to enlarge three specific strands (chains of interleaved data and parity blocks) determined by the algorithm. The pattern is repetitive, i.e. the new parity block is used together with a newcomer block as input for the next XOR operation. Hence, the algorithm propagates redundant information in regular patterns into disks arrays. The code is also known as p-HEC because its p double-helix strands that resemble a DNA topology. The code builds, in total, 2 horizontal and $2 \times p$ helical strands. In a short paper, we gave some insights into the complex calculation of reliability in entangled storage systems by analysing 5-HEC reliability with a hierarchical decomposition of serial and parallel subsystem [17]. Recently, we proposed a simpler algorithm, which merely uses one single strand, showed that entanglements can provide more fault tolerance than mirroring arrays [16]. The early evidence of high reliability in entanglement codes is promising. Hence, it has motivated our study to answer important questions about redundancy propagation and its impact on a storage system.

Alpha entanglement codes provide a general model and unifying theory for the design of practical entanglement codes. We define α-entanglement families and evaluate them in detail. This work significantly extends previous studies on entanglement codes, which become particular entanglement cases of AE codes. The previous defined p-HEC method corresponds to the family of triple-entanglements with $s = 2$, in short AE$(3, 2, p)$. Our motivation is to design practical erasure codes for large scale environments instead of another code based on traditional erasure codes originally designed for data transmission over erasure channels. We believe that, as indicated by other authors [27], the translation of communication channel into a storage device is a simplification that hides maintenance costs in long-term storage. This awareness motivated this work that investigates further on entanglement models and their implementation to build an entangled storage system.

III. Alpha Entanglement Codes

α-entanglements(s, p) is a family of erasure codes to tangle data and redundant blocks with the goal of increasing the scope of redundant information by means of propagation. The encoder builds block chains, strands, that alternate data and redundant blocks. The entanglement function computes the exclusive-or (XOR) of two consecutive blocks at the head of a strand and inserts the output adjacent to the last block. The strands are intertwined creating a mesh of entangled blocks.

A. What Is Redundancy Propagation? Why We Should Care?

Our notion of redundancy propagation challenges the traditional notion of redundancy. In engineering, the redundancy of a system is explained in terms of how the components are arranged in serial and parallel combinations. If elements are

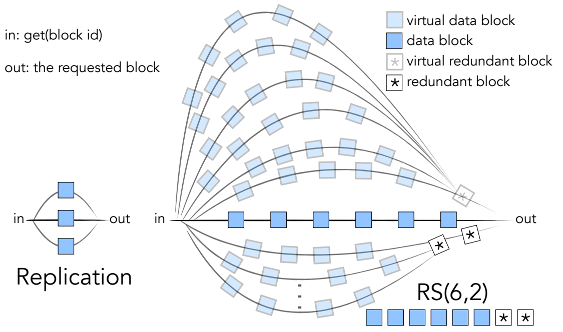

Figure 1: Replication creates parallel paths, any of them can satisfy the request. RS codes create parallel and serial paths, e.g., RS(6,2) splits the source in 6 blocks and expands it with 2 redundant blocks. Any combination of 6-out-of-8 blocks is a valid output. Parallel paths are constructed with virtual blocks.

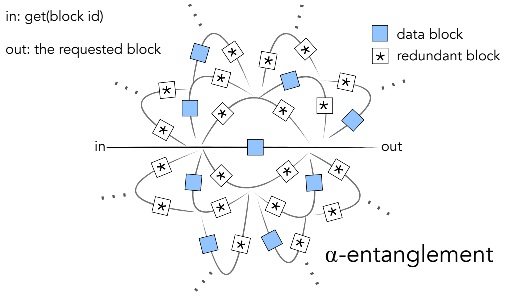

Figure 2: AE codes create concentric/parallel paths for each data block. Any path that connects in-out is valid to read the central block. Paths that are closer to the center have less elements in serial combinations while paths that are further away require more blocks but increase the chances of success.

arranged in series, it means that the system needs all elements to satisfactorily perform the task. Elements that are arranged in parallel increase the chances that the system will operate successfully. Indeed, an encoder algorithm creates paths that connect data blocks in serial and parallel arrangements. A decoder algorithm uses the paths to decode data.

Fig. 1 illustrates the classic methods to store data redundantly in a system. Replication improves reliability by creating n parallel paths of single blocks. RS codes and RAID-like codes improve performance by using serial paths of k blocks. They also improve reliability by creating virtual parallel paths since any combination of k-out-of-n blocks, with $n = k + m$, are useful to read the k data blocks. The paths are virtual since blocks are not replicated, hence, this method is storage efficient. Though, it is not optimal in terms of bandwidth and I/O operations, in particular for single failure repairs.

Fig. 2 illustrates the concept of redundancy propagation. Any arbitrary data block propagates redundant information to other blocks. The figure shows many valid paths to read the block located at the center of the image. The block is at the center of its loosely called *propagation sphere*. The entanglement codes create, with a few simple rules, an arrangement of overlapping-spheres packing (each data block shown in the figure is at the core of its own propagation sphere, which are not shown for simplicity). In spite of the overlapping, each data block is uniquely identified by its position in a lattice, which determines unique combinations of two or more blocks that are useful to read the block. The entanglement process creates concentric paths for each data block. The paths that are close to the center have less elements in serial combinations while the paths that are more distant require more blocks. The decoder uses the shortest available path to repair a missing data block. It rarely needs to use a long path since single failures are efficiently repaired using two blocks. Since the decoder can repair multiple single failures in parallel, a long path affected by two or more failures may be repaired in a single round. Otherwise, the decoder tries again in the next round. Redundancy propagation increases the probability of successfully recovering from catastrophic failures. It also makes difficult tampering with data because information is propagated to many elements in the system.

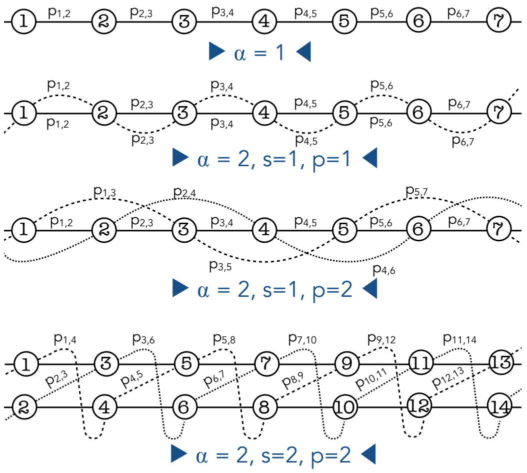
1 2 3 4 5 6 7
$p_{1,2}$ $p_{2,3}$ $p_{3,4}$ $p_{4,5}$ $p_{5,6}$ $p_{6,7}$
α = 1
α = 2, s=1, p=1
$p_{1,3}$ $p_{2,4}$ $p_{3,5}$ $p_{4,6}$ $p_{5,7}$
α = 2, s=1, p=2
8 9 10 11 12 13 14
$p_{1,4}$ $p_{2,3}$ $p_{3,6}$ $p_{4,5}$ $p_{5,8}$ $p_{6,7}$ $p_{7,10}$ $p_{8,9}$ $p_{9,12}$ $p_{10,11}$ $p_{11,14}$ $p_{12,13}$
α = 2, s=2, p=2

Figure 3: *Single- and double-entanglements.* A node, d_i, represents a data block and an edge, $p_{i,j}$, represents a parity block. Nodes are added to the graph in sequential order. They are uniquely identified by its position i, indicated inside the nodes.

B. How Do We Achieve Redundancy Propagation?

AE codes are a family of codes composed by: (i) Single entanglements or *1-entanglements*, $\alpha = 1$, built with a single horizontal strand. (ii) Double entanglements or *2-entanglements(s,p)*, $\alpha = 2$, built with s horizontal strands and one class of p helical strands. (iii) Triple entanglements or *3-entanglements(s,p)*, $\alpha = 3$, built with s horizontal strands and two classes of p helical strands. (iv) n-Tuple entanglements or *n-entanglements(s,p)*, $\alpha = n$, built with s horizontal strands and $n-1$ classes of p helical strands. Fig. 3 shows examples. A node, d_i, represents a data block and an edge, $p_{i,j}$, represents a parity block. The entanglement creates new parities and builds strands by adding elements sequentially. Strands are intertwined forming a graph. A node is uniquely identified by its position i in the graph. An edge $p_{i,j}$ connects nodes d_i and d_j. It represents a parity block that is computed by XORing the last parity and data block of the same strand.

Strands. Each individual strand corresponds to a single entanglement. The complex combination of strands brings additional properties and increases fault tolerance. One of the emergent properties of α-entanglements is that failure patterns that are not tolerated with single entanglements become innocuous in entanglements with $\alpha > 1$ since data blocks are reconstructed through any of the α strands in which they participate. We define three classes of strands, the horizontal (H), the right-handed (RH) and the left- handed (LH) strands. In a 3D space, the helical strands revolve around a central horizontal axis and grow towards the right direction. This work considers helical strands that connect horizontal strands with a diagonal of slope 1. If possible, there is a balanced quantity of RH and LH strands. In other words, 2-entanglements are composed with one class of helical strands (RH or LH) and 3-entanglements are composed with both classes of strands.

A lattice is composed by s horizontal strands, and $\alpha - 1$ helical strand classes. Hence, the total number of strands is given by the formula $s + (\alpha - 1) \cdot p$.

Code Parameters. Data propagation is tailored with three parameters: α, s, p. The parameter α specifies the number of parities created per data block. It also describes the number of strands in which one data block participates. Tuning α impacts on the resilience of the structure. More parities create more redundancy, on the other side, the storage overhead increases. The parameter α determines the code rate, which is computed with $\frac{1}{\alpha+1}$. Optionally, systems that only store parities have an improved rate of $\frac{1}{\alpha}$. This work focuses on codes with $\alpha \in [1, 3]$ and gives some guidelines for large values of α. The parameter p specifies the number of helical strands, and the parameter s defines the number of horizontal strands. Increasing s and/or p impacts positively on the structure's resilience without adding storage overhead.

The parameter α and the lattice geometry put some constraints on the parameters. 1-entanglements are formed with only one chain of entanglements, therefore, $s = 1$ and $p = 0$. α-entanglements with $\alpha > 1$ are valid when $p \geq s$. An invalid setting, i.e. $p < s$, causes a deformed lattice.

Code Specification. The encoder constructs a helical lattice using data and parity blocks with identical size. Data blocks are represented by nodes, and parities are represented by edges. Each node belongs to α strands, each edge belongs to only one strand. There are three categories: top, central and bottom nodes that determine the encoder rule. The encoder processes data blocks iteratively and keeps a counter c that indicates the last processed block. First, a new data block is assigned an index $i = c+1$ that indicates its lattice's position. We refer to this data block as d_i. Second, the node category is determined by: *top* if $i \equiv 1 \mod s$, *bottom* if $i \equiv 0 \mod s$, and *central* if $i > 1 \mod s$. Third, the encoder computes α parities for block d_i. Each new parity $p_{i,j}$ is computed by XORing the data block d_i with an existing parity block $p_{h,i}$, whose index h and j are determined using rules tables. Fig. 4 shows a lattice created with AE(3,5,5) using the rules given in Tables I and II. The lattice is composed by 15 strands: 5 H strands H_{1-5}, 5 RH strands RH_{1-5}, and 5 LH strands LH_{1-5}. The decoder repairs a node using two adjacent edges that belong to the same strand, thus, there are α options. To illustrate, to repair d_{26} from Fig. 4 using H strand, the decoder computes the XOR($p_{21,26}$,$p_{26,31}$), whose indexes are determined with Tables I and II respectively. The decoder repairs an edge using any of the two incident nodes on the damaged edge and its corresponding adjacent edge, hence, there are always two options. For instance, using the same figure, to repair $p_{21,26}$, it computes the XOR(d_{21},$p_{16,21}$).

TABLE I: 3-entanglement INPUT rules. AE(3,5,5) example: on RH strand top node d_{26} is tangled with $p_{25,26}$

	d_i is tangled with $p_{h,i}$, and h index is		
d_i **position**	**H strand**	**RH strand**	**LH strand**
top	$i - s$	$i - s \cdot p + (s^2 - 1)$	$i - (s - 1)$
central	$i - s$	$i - (s + 1)$	$i - (s - 1)$
bottom	$i - s$	$i - (s + 1)$	$i - s \cdot p + (s - 1)^2$

TABLE II: 3-entanglement OUTPUT rules. AE(3,5,5) example: on RH strand top node d_{26} entanglement creates $p_{26,32}$

	d_i entanglement creates $p_{i,j}$, and j index is		
d_i **position**	**H strand**	**RH strand**	**LH strand**
top	$i + s$	$i + s + 1$	$i + s \cdot p - (s - 1)^2$
central	$i + s$	$i + s + 1$	$i + s - 1$
bottom	$i + s$	$i + s \cdot p - (s^2 - 1)$	$i + s - 1$

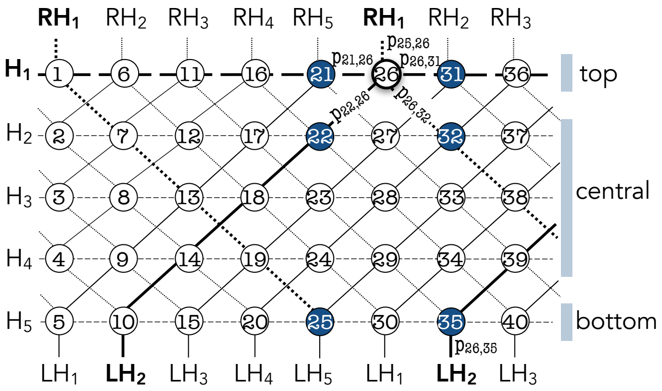

Figure 4: AE(3,5,5) lattice with s=5 (rows) and p=5 (columns/diagonals). Any node d_i participates in $\alpha = 3$ strands, e.g. d_{26} is a top node that belongs to H_1, RH_1 and LH_2 strands. Colored nodes are at one hop of node d_{26}.

Implementation Details. The lattice acts as a virtual layer on top of the storage devices, or storage nodes in a p2p application. As with any other redundancy method, storage systems use mapping algorithms to store and locate encoded blocks according a placement policy and the available resources. The encoding and decoding implementation may use a client-based, middleware-based, or backend-based approach.

Reducing Storage Overhead. The storage overhead does not grow gradually, i.e., increasing α in one unit means that the storage overhead increases in steps of 100%. We propose two strategies to enhance the code rate. A first option is to start with a low α and increase the value later as required. A second option is to puncture the code. Puncturing is a standard technique use in coding theory in which, after encoding, some of the parities are not stored in the system. We will report our findings on this subject in the near future.

Anti-tampering Property. The protection against data manipulation is an emergent property. To go undetected, an attacker should modify the α strands in which the targeted block participates by replacing all the parities computed from its position to the closest strand extremity. For example, to tamper d_{26} in AE(3,5,5) shown in Fig. 4, the attacker needs to recompute $d_{26,31}$, $d_{31,36}$ and all the parities on the strand until the end of H_1 and do the same for RH_1 and LH_2.

IV. Entangled Storage System

In this section, we explain how AE codes can be adapted to diverse applications without being restricted to a particular storage architecture. We describe two cases: (i) a cooperative storage system built on top of a decentralised database, and (ii) a centralised storage architecture (disk arrays).

A. Use Case: A Geo-Replicated Backup

A community creates a cooperative storage network to share storage and bandwidth resources. Users keep their own data in their local computers (nodes) and upload redundant information to geographically distributed nodes. The storage capacity used by each user has to be agreed at the beginning and renegotiated as needed. The system is a two-tiered architecture that aims at supporting efficient data protection and integrity. The lower tier is composed of *storage nodes* that share space to store parity blocks from other users. The upper tier consists of *broker nodes* that encode and decode data. In this example a single node has both roles a broker and storage node. The nodes are organized in a loosely connected cluster. In order to avoid confusion with the helical lattice graph, the text clarifies if “node” refers to a graph vertex; otherwise we use the terms *d-block* for a vertex and *p-block* for an edge. It is worth noticing that the final design requires consideration of various types of attacks such as free riding abuses that are beyond the scope of this work. In particular, methods to defend against free riding were proposed previously by other authors [28].

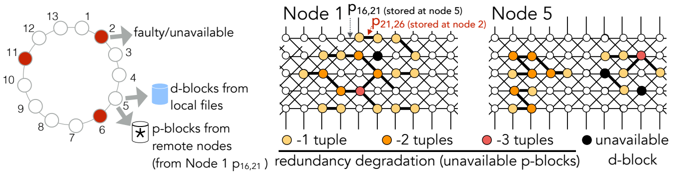
13
1
12
2
faulty/unavailable
11
3
4
10
d-blocks from local files
5
9
8
7
6
p-blocks from remote nodes (from Node 1 $p_{16,21}$)
Node 1
$p_{16,21}$ (stored at node 5)
$p_{21,26}$ (stored at node 2)
Node 5
-1 tuple
-2 tuples
-3 tuples
unavailable d-block
redundancy degradation (unavailable p-blocks)

Figure 5: *Failure mode.* In a healthy AE(3,5,5) lattice each d-block has three pp-tuples stored at remote nodes. Unavailable nodes degrade redundancy. Each lattice is affected differently.

TABLE III: Node 1 recomputes $p_{21,26}$ after failure.

Steps	**Example**
1. Obtain dp-tuple id:	$\{key_{21}, key_{16,21}\}$,$\{key_{26}, key_{26,31}\}$
2. Choose p-block id:	$key_{16,21}$
3. Compute location key:	n_5
4. Get block:	$p_{16,21}$
5. Repair block:	$p_{21,26}$

Redundancy Scheme. Each user manages his own entanglement lattice via the broker, hence multiple lattices coexist in the system. Moreover, the system could keep lattices with different settings. In the simplest scenario, the broker is a service running in the user’s computer. In more complex scenarios, the broker could be a super node that encodes data on behalf a group of users. The broker recognizes files for backup. First, it prepares the file by splitting it into d-blocks. To entangle them, it needs to fetch some parities (p-blocks) from remote nodes. For performance, it can keep the parities from previous encodings in memory. In that case, the memory footprint of the broker is linear in the number of distinct strands that create the entanglement lattice. For instance, AE(3,5,5) requires to keep in memory the last p-block of its 15 strands. If the broker crashes, it only needs to retrieve the p-blocks from the remote nodes. Blocks are located by their key, which can be a value derived from the node id and the block position in the lattice (such as a hash of both values). Parities are mapped to the storage nodes using a deterministic or random placement algorithm. In a failure-free environment, users can access their data directly from their local computers. In other words, decoding is not required.

Failure Mode. Repairs occur when users do not have access to their local d-blocks or when their lattices deteriorate due to faulty or unavailable nodes. How node failures are detected and notified is not discussed here but there is plenty of literature on this topic. Fig. 5 shows a scenario in which three nodes are unavailable and illustrates how the incident impacts on multiple elements in distinct lattices. The size and characteristics of the incident dictate the patterns formed on the lattices. Parity block repair is automatically distributed, assuming that all users will be interested in the regeneration of their lattices to maintain the same level of redundancy for their data. If a node is not able to repair the lattice, other nodes can do repairs on their behalf as well.

Repairing d-blocks requires complete pp-tuples (two p-blocks) and repairing p-blocks requires complete dp-tuples (one d-block and one p-block). Fig. 5 indicates redundancy degradation. Some nodes have incomplete tuples. Each unavailable block generates a single-failure repair. Table III enumerates the steps to regenerate the parities located in faulty nodes and illustrates with an example. Node 1 performs the steps 1-3 and 5 locally, and node 5 answers step 4.

B. Use Case: Disk Arrays

Disk arrays can increase the performance of a system because individual requests are served faster when multiple disks collaborate to increase throughput. In addition, the possibility to answer multiple requests in parallel gives the chance to increase the system’s I/Os rate. Some form of redundancy is required to compensate the increase of combined disk failure rate when writing to disk arrays. RAID organizations [1] are a well-established solution for the industry and the home user to increase the performance and reliability of large arrays of inexpensive disks. Although there is still future for cheap spinning disks in data centres and home solutions, the market changed substantially in the last decades.

Storage at exabyte scale brings difficult challenges: (i) *Rebuild time* may take hours or days. Systems that tolerate 2 failures dominate the market. More failures during rebuilds are a source for data loss. (ii) The *disparity between technology development* means that disk capacity has been grown steadily while bandwidth lays behind. (iii) *Cloud-based storage has received more research attention than RAID solutions*, even though RAID is still in use for many reasons, e.g. privacy. (iv) *Failure rate* is mostly related to system's size [29]. (v) *Failure correlation* is frequently disregarded. The assumption that failures are independent and that time between failures is exponentially distributed is not valid [30].

We propose to rethink disk arrays with the use of AE codes. The acronym RAID states for redundant arrays of independent[1] disks. Given that AE codes creates interdependencies between devices, it makes sense to drop the term "independent" and replace it with "interdependent". Note that this use case is valid for log-structured append-only storage systems.

1) Entangled Mirror: In earlier work, we proposed two different array organizations based on simple entanglements [16]. Both organizations require equal numbers of data and parity drives; therefore, the array has the same space overhead as mirroring. For the sake of completeness, a quick recap is included in this subsection.

Full Partition. In this approach, a node represents a data drive and an edge represents a parity drive. Blocks are written sequentially on the same drive type. The process does not spread content across drives. Most of the drives will remain idle and can be powered off as in a massive arrays of idle disks, MAID, configuration [31], which could result in significant energy savings.

Block-level Striping. This second approach distributes data over all available drives to improve performance.

In an entanglement chain, blocks that are located at the extremities have less redundancy. This problem has more impact on full partition where the content that is at extremity of the chain is equivalent to the stored data in one disk. At block-level striping, the amount of data is equivalent to the block size. We proposed open and closed chains, the second approach to address this problem. We showed that in full partitions both approaches provide better 5-year reliability than mirroring, reducing the probability of data loss by respectively 90 and 98 percent.

2) RAID-AE: Storing data in disk arrays is often held as an outdated solution. Given the scalability issues of RAID and the growing cloud market, some organizations move data to the cloud. However, in-house storage is still used by many industries, e.g. finance market, government, and research institutions. On the other side, many cloud storage services are built on infrastructures that use RAID-like storage solutions. For example, Backblaze's backup service uses a RAID system built on top of Linux and RS codes [32].

There are three ways to increase the redundancy of data when configuring disk arrays: mirroring (discussed previously), parities and erasure codes. RS codes became a sort of de-facto industry standard for erasure coding, and particularly to archive data. Perhaps the main reason for their broadly acceptance is because, when the storage market became to grow, they were already an established solution for digital communications. In brief, RS codes are storage efficient, well-understood, and open-source libraries are available. But the bandwidth and I/O cost during the repair process is only partially solved with optimal locally repairable codes [11].

We briefly discuss the properties of a RAID built with our entangled-based solution.

Scalability. RAID-AE can make improvements for both horizontal and vertical scalability. In a RAID-AE array, it is possible to add more disks and change the storage capacity by replacing disks or creating volumes. Both actions may be done dynamically without interrupting the service and without encoding data again.

Never-ending Stripe. Entanglement codes change the classical notion of *stripe* due to the possibility for writing on a never-ending stripe (a lattice has no limits), on the other side, the write penalty is $\alpha + 1$. For clear examples, we use RAID5 but with more complex RAID organizations the situation remains the same. The way to compute parities is based on a fixed-width stripe, e.g. in a 6+1 disk RAID5, one parity unit is computed using 6 data units. When one disk is added to the array, the new array 7+1 disk RAID5 requires re-encoding parities, one parity unit is now computed using 7 data units. Additionally, there is a large penalty when a disk fails because of the bandwidth and I/O overhead to repair each missing unit. Parity declustering [33] reduces the rebuild time by distributing repairs across devices in a large array but this technique cannot reduce the overhead mentioned above. We argue that the scope for major improvements is limited by the stripe size, increasing the length of a stripe has a prohibitive effect in repairing data.

Degraded Reads. The stripe size impacts on degraded reads in RAID-like solutions. In data centre environments where disks are distributed in different machines and racks, the performance of degraded reads matters. RAID-AE provides many alternative paths to read data that is temporarily unavailable due to software updates, schedule restarts, etc.

Other Features. RAID-AE can be implemented to provide distributed repairs and load balance for read intensive workloads. One important difference in our model is that we do not assume failure independence and because the parameter α can change in future, the system can scale in fault tolerance.

In sum, RAID-AE will permit different arrangements with

[1] Patterson et al. used the word "inexpensive" but solutions became complex and expensive and "independent" was gradually adopted.

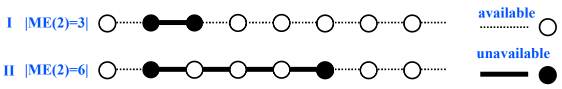

Figure 6: *Primitive forms.* Single entanglements cannot tolerate I) triple failures affecting two adjacent nodes and their incident shared edge and II) its extended form in which the nodes are not adjacent but all the connecting edges are unavailable.

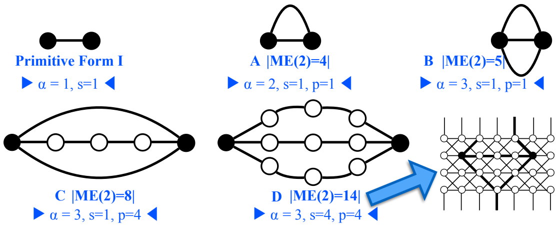

Figure 7: *Complex forms.* When $\alpha \geq 2$ primitive forms do not cause data loss only their combinations into complex forms as shown in patterns A-D.

trade-offs between capacity overhead, network bandwidth overhead, rebuild time and fault tolerance.

V. Evaluation

We examine the design of AE codes to understand how the code settings impact on fault tolerance and write performance. Then, we compare AE codes with traditional redundancy schemes (RS codes and replication) in disaster recovery by measuring data loss and vulnerable data. RS codes conceptualize the idea of an "ideal code" that has optimal characteristics and for that reason can be used as a baseline to compare with other codes [34]. In addition, RS codes are well-understood and many other codes are built on top of them. Given the radical differences of AE codes with other codes the comparison is not a trivial task. AE codes construct a scalable fabric of interdependent data that propagates redundancy, and this overlay can be mapped to the physical storage layer. To the best of our knowledge, our notion of redundancy propagation has never been explored at this level in other codes. Oversimplification of our model would lead to unfair results and adding more codes in the comparison would create unnecessary complications. Instead, we design metrics that focus on the main characteristics presented in the previous sections to study them from different angles. To complete this study, we draw attention to the hidden role of maintenance and the risk of data loss in unreliable environments. We investigate how much redundant data are needed in unreliable environments. Finally, we show evidence on the impact of data placements for different redundancy schemes.

A. Code Parameters and Fault Tolerance

We have conducted a large study on the erasure patterns that a helical lattice cannot tolerate. The motivation was to understand how the code parameters impact on the fault tolerance. More specifically, we show visual evidence of how the parameters s and p increase fault tolerance without the need of additional storage or bandwidth.

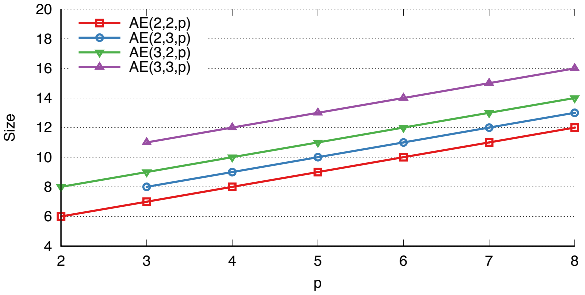

Figure 8: $|\mathbf{ME}(2)|$ increases with larger s and p.

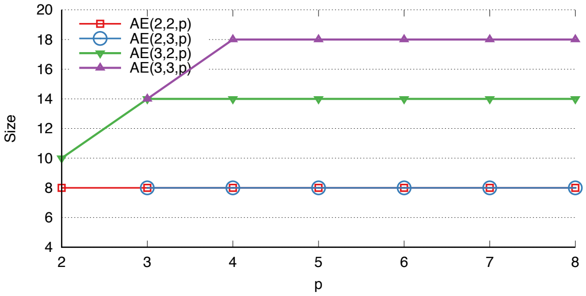

Figure 9: $|\mathbf{ME}(4)|$ remains constant for $\alpha = 2$, and increases with s for $\alpha = 3$.

A minimal erasure (ME) is an irreducible pattern that causes the loss of data and parity blocks. These patterns are irreducible in the sense that the removal of any of their blocks enables the possibility to recover some of the erased blocks. The MEL [19] is the enumeration of all minimal erasures in irregular XOR-based flat codes. We use a variation of Wiley's study to gain more knowledge about the pattern size and its impact on data block loss. Wiley focused on characterizing failure patterns by their size without making the distinction between pattern size and actual data loss, or the impact that code parameters have in those failure patterns. Our goal is to characterize minimal erasure patterns by their size and amount of data loss. Our modification includes the notion that an erasure pattern of size y blocks only has x data blocks, with $y > x$. *Ideally, we want patterns with $y \gg x$ since that means a high fault-tolerance, which decreases the probability that the decoder fails, but when it fails only a small fraction of blocks are data blocks.* For many patterns we can increase the ratio y/x using the code parameters. An outstanding characteristic is that the parameters s and p permit increasing the ratio y/x without generating more storage overhead or increasing the repair cost of single failures. The increase in fault tolerance when tuning the parameters is evidenced by a cross-study comparison of the size of minimal erasure patterns for different code parameters. This study does not identify all erasure patterns for α-entanglements. To minimize the burden of such task, we concentrate only on the most relevant patterns to determine which of them have lower and upper bounds for the redundancy propagation achieved by entanglement codes. Fig. 6 and 7 present examples for ME(2). Our findings come from visual inspections and from verifications conducted with a tool[2] implemented in the Prolog language.

Results. Fig. 8 and 9 show that $|\text{ME}(x)|$ is minimal when $s = p$, i.e. the code provides the minimum possible fault-tolerance for a given α and s. Although the code can provide more fault-tolerance when p increases, the setting $s = p$ cannot be judge as good or bad per se. Trade-offs between performance and reliability should be considered to choose the best parameters. Fig. 9 shows the impact of α on patterns $\text{ME}(2^{\alpha})$. In this case, the parameters s and p have insignificant impact on fault tolerance. The reason behind this behaviour is that with $\alpha = 2$ redundancy is propagated across elements that form a square pattern (4 nodes and 4 edges, hence $|\text{ME}(4)| = 8$) and these elements cannot be repaired if all are lost. With $\alpha = 3$, $|ME(4)|$ becomes larger with s but not with p, in this dimension, redundancy is propagated across a cube pattern, hence $|\text{ME}(8)| = 20$ for AE(3,3,3), not shown for space reasons.

Beyond $\alpha = 3$. We are still investigating entanglement codes with $\alpha > 3$. We can safely speculate that the fault-tolerance would improve substantially. We expect to find upper bounds for fault tolerance defined by the size of an n-hypercube. This upper bound has impact on $|\text{ME}(x)|$ with $x = 2^{\alpha}$, for example, for $x = 16$ redundancy propagation would be enclosed in a tesseract (4-cube). In addition, for larger α values we expect improvements in repair performance. However, it is not clear how to connect the extra helical strands. In double- and triple-entanglements the RH and LH strands are defined along a diagonal of slope 1. One possible option is to add additional helical strands with a different slope.

B. Code Parameters and Write Performance

The values of parameters s and p impact on the number of data blocks that need to wait to be entangled. When $s = p$, this number is maximized and entanglements can be done in parallel operations, see Fig. 10. A *sealed bucket* contains a data block and the α parities created by the entanglement process. A data block can be fully entangled when the α input parities needed in the process are already computed and kept in memory. The memory requirement for full-writes is $O(N)$, where N is the number of parities computed in the full-write.

C. Simulations

We run simulations to understand the scalable behavior of AE codes, assess them against catastrophic failures, compare them with (k, m)-codes and address the question of how much redundancy is needed to keep data safe in unreliable environments that have minimal (or zero) maintenance, i.e., repairs and efforts to restore redundancy after failures.

Examples of Unreliable Environments. An unreliable environment could be a peer-to-peer (p2p) network where nodes join and leave frequently. Redundancy helps to protect data from real node departures and for maintaining high availability when hosts are offline. A relevant work on p2p storage showed a large-scale cooperative storage is limited by unreasonable cross-system bandwidth [35]. Although this study from 2003 has some results that need update like current hardware trends and Internet connections, there are interesting insights from real world p2p networks. Nodes availability in a p2p network is very variable and maintenance swallows up most of the node's resources. Another relevant remark from the same work is that a system needs up to 120 copies to achieve high availability (6 nines) using replication, while erasure codes reach 6 nines with only 15 times the original storage needs. In archival storage systems, data durability is an endeavour that depends on the engineering aspects of the system but on the economics too [36], [37]. From the perspective of data centers, hard disks are unreliable components that contribute to high maintenance cost. To give a rough idea, the annual cost due to hardware

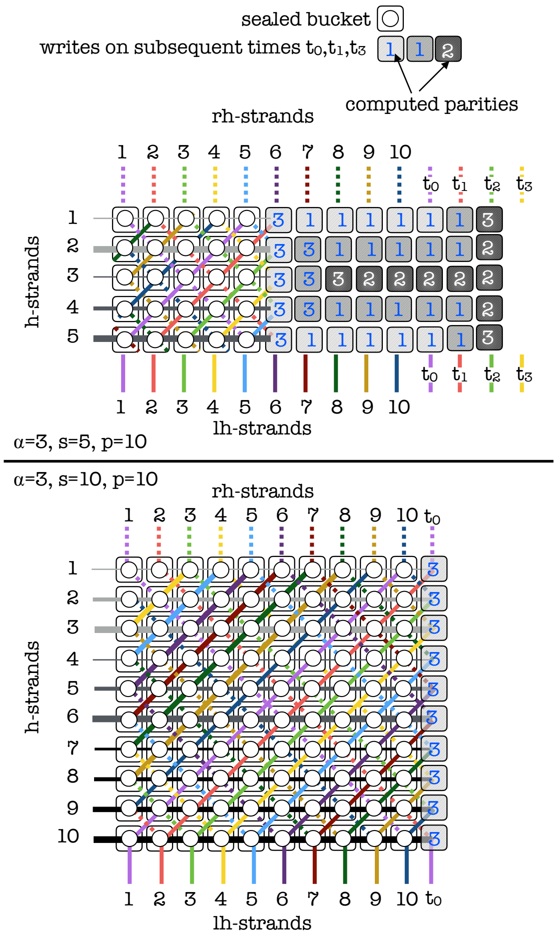

Figure 10: Write performance for $p > s$ and $s = p$. Full-writes are optimized when $s = p$ since all the parities needed to complete triple-entanglements and seal the buckets are available in memory. When $p > s$, one option is to do full-writes for s elements, a second option is to write buckets partially using the available parities.

[2] Available at request.

repairs (mostly due to hard disks failures) for a data center with more than 100,000 servers was estimated to be over a million dollar by Microsoft's researchers in 2010 [38].

Selected Redundancy Schemes. We study 4 different settings for Reed-Solomon codes and 3 different settings for alpha entanglement codes, see Table IV. In addition, we compare up to 4-way replication since 300% is the maximum additional storage considered in this paper. Replication requires much more storage to offer high fault tolerance but it does not have overheads for single failures. This evaluation substantially improves previous evaluations [17] since we include many more code settings, millions of blocks and placements.

Simulation Environment. We investigate what happens with data in environments that either accumulate plenty of failures before repairs take place or a large number of failures happen all at once. The metrics described in this section follows the criteria used in an on-going project intended to help in the comparison of codes [39]. All experiments are done with 1 million data blocks and the corresponding number of encoded blocks for each code setting, e.g. RS(10,4) generates 400,000 encoded blocks while RS(8,2) generates 250,000 encoded blocks. That means that the number of stripes is different for each code setting, e.g. RS(8,2) creates 125,000 stripes, and RS(5,5) creates 200,000 stripes. Blocks are synthetically generated and stored in tables adapted for each code method. Table V shows a simplified version of the AE table that uses three columns to identify the block, one column to determine the location, one column to determine the availability of the block and one final column to specify if the block is repaired.

Block Placements. Blocks are distributed in n locations using random placements, i.e., each block is assigned a random number from 0 to $n-1$. We present the results of simulations for medium size ($n = 100$) storage system For example, the 1 million data blocks and 0.4 million encoded blocks created with RS(10,4) are distributed to 100 locations with a mean of 14,000 blocks per site and a standard deviation $\sigma = 130.88$. When locations are selected at random, it can happen that blocks from the same RS stripe are assigned to the same location. In our previous example, in a total of 100,000 stripes, only 38,429 had their 14 blocks distributed to different locations. The rest of the stripes were distributed in locations (stripes): 8 (5), 9 (39), 10 (475), 11 (3,746), 12 (17,076), 13 (40,230). As expected, blocks had a fairly balanced distribution by increasing n, e.g., 91,167 stripes had their 14 blocks in different locations with $n = 1,000$. Spreading blocks evenly over nodes is a well-known scalability problem that affect load balance and system's reliability. While, at first sight, the distribution of RS encoded blocks is not even for a medium size system, we have run other simulations with a larger number of distinct locations and the comparisons remain close to the ones presented here. An important remark is that the random distribution of alpha entanglement encoded blocks in a medium size system affects the code performance in a worse way. For AE(3,2,5), the equivalent of 5-HEC, a lattice section of 80 elements (data and parities) need to be distributed in different failure domains. The number was obtained from previous work [17], where we assumed a round robin placement policy. This requirement assures that neighbour elements have more chances to be available in repairs that involved multiple blocks. We think a round robin placement might be difficult to implement. Therefore, this study answers what happens if we use random placements. It is expected that the subset of 80 blocks will not be fairly spread over all 100 locations, but does it affect the ability of the code to recover from disasters?

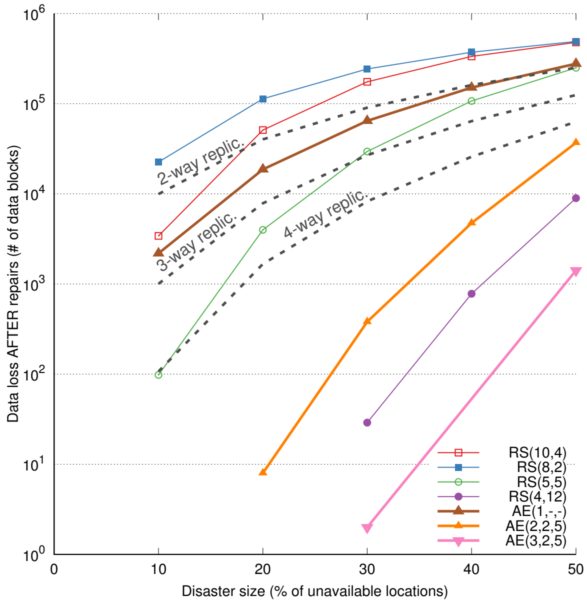

Figure 11: Data blocks that the decoder failed to repair.

Disaster Recovery. The framework simulates disasters by changing the availability of a certain number of locations (10-50%) and trying to repair the missing data blocks. When locations are unavailable, e.g., due to a faulty storage device, the repair process has to handle a very large number of single failures since a single location failure cause damage to multiple positions in the helical lattice.

1) Metric: Data Loss: This metric only counts data blocks that satisfy the following conditions: a) its location is unavailable, b) the repair process was unsuccessful. Other available data blocks that belong to damaged stripes are not counted as lost. A RS stripe is damaged when more than m blocks are unavailable. For example, a RS(8,2) stripe that has one encoded block and two data blocks in unavailable locations only contributes to the metric with two data blocks and the other six data blocks are considered available. In real-world implementations, it is probably that a RS damaged stripe cause more data loss than the one indicated with our metric. In AE codes, each block is encoded individually and data loss counts exactly how many data blocks are lost. Fig. 11 shows that AE(3,2,5) outperforms RS(4,12) even though both have the same storage overhead. Data loss for AE(1,-,-) is one order more than RS(5,5), a code that uses the same storage overhead, but the gap between both curves decreases when the number

TABLE IV: Redundancy schemes (examples for trivial replication cases not shown). AS: additional storage, SF: single failures

Cost	**RS(k,m)**	**AE(α,s,p)**	**n-way replication**	**RS(10,4)**	**RS(8,2)**	**RS(5,5)**	**RS(4,12)**	**AE(1,-,-)**	**AE(2,2,5)**	**AE(3,2,5)**
AS	$\frac{m}{k}\cdot 100\%$	$\alpha\cdot 100\%$	$(n-1)\cdot 100\%$	40%	25%	100%	300%	100%	200%	300%
SF	k	2	1	10	8	5	4	2	2	2

TABLE V: AE: d-block d_{26} and p-blocks $p_{21,26}, p_{26,31}$, $p_{22,26}, p_{26,35}$, $p_{25,26}, p_{26,32}$. Locations 3,12,47,56 are unavailable. Block d_{26} is repaired via RH strand's p-blocks.

i	**j**	**Type/Strand**	**Location**	**Available**	**Repaired**
26	26	d	56	FALSE	**TRUE**
21	26	h	3	FALSE	TRUE
26	31	h	47	FALSE	FALSE
22	26	lh	12	FALSE	FALSE
26	35	lh	28	TRUE	FALSE
25	26	rh	91	**TRUE**	FALSE
26	32	rh	39	**TRUE**	FALSE

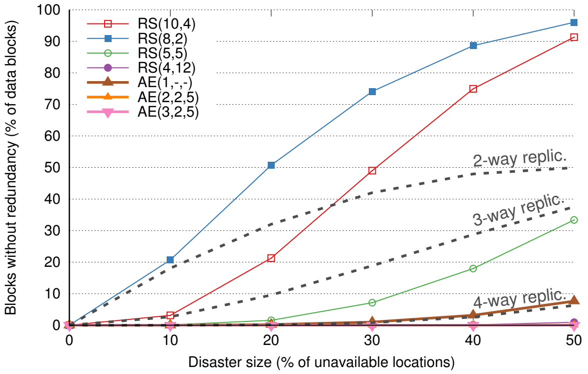

Figure 12: Data blocks without redundancy.

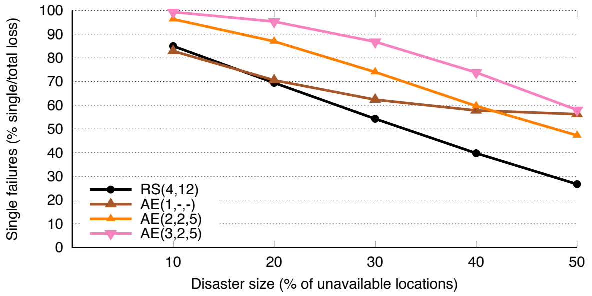

Figure 13: What part of repairs are single failure repairs?

of unavailable locations increases. In fact, the RS(5,5) curve shows an interesting behavior of RS codes. When the disaster size is relatively small (10%), data loss of RS(5,5) is equivalent to 4-way replication. But the quality of RS declines with larger disasters, i.e., with 30% of unavailable locations data loss of RS(5,5) is equivalent to 3-way replication and with 50% of unavailable locations RS(5,5) causes the same data loss as 2-way replication. AE(2,2,5) excel at repairing blocks and the storage requirement is equivalent to using 3-way replication.

2) Metric: Vulnerable Data: This metric represents the total number of data blocks that are not protected by any other redundant block after the repair process finishes data recovering. It illustrates how the level of redundancy decays when only minimal maintenance operations are done in the system. Minimal maintenance happens when the decoder repairs unavailable data blocks but make no attempts to repair unavailable parities. However, some parities are repaired if they are part of the same stripe of an unavailable data block. A storage system can receive little maintenance under the following common scenarios: data block repairs are given priority, there is a lack of incentive to recover parities, missing parities may be difficult to detect when repairs are triggered by data block read failures. Minimal maintenance can pose a threat to a big portion of the data in a storage system. Fig. 12 shows the high percentage of data blocks that remain without redundancy after blocks repairs with RS codes. We observe that RS(5,5) performs worse than AE(1,-,-) and leaves more data without redundancy when failures affect more than 20% of the locations. This result explains why in Fig. 11 the gap between the data loss curves for both codes becomes closer in large disaster scenarios. The RS(4,12) code is the only comparable to the high protection provided by AE codes. Our results are somehow related with the impact of *combinatorial effects* and *peer availability effects* mentioned by other authors [40]. These two effects explain the factors that benefit erasure codes and replication codes in p2p systems. Erasure codes take benefit of the combinatorial effect, however, they are more dependent of the availability of multiple locations. The authors added that when the peer availability is low, replication can be better than erasure codes. Alpha entanglements are less affected by the peer availability effect since only two blocks are used for repairs but they profit from the combinatorial effect as illustrated in Fig. 2.

3) Metric: The Cost of Single Failures: In (k,m)-codes, repairing single failures (SF) is expensive. SF require k times bandwidth overhead and I/O operations at k locations. Alpha entanglement codes handle single failures with a radically different approach. SF are repaired with a fixed "k=2" (two parity blocks) for any code setting. However, repairing the full system may require several rounds. Fig. 13 shows the percentage of data blocks that required SF repairs. For AE codes, we computed the proportion between all single failures solved at the first round and the total data loss repaired in the system. For reference, we compute the same proportion (without considering rounds) for RS(4,12), which is the code with highest locality among our selected RS codes and superior than other locally repairable codes like the HDFS-Xorbas implementation [14]. For RS codes, the repair efficiency is very bad for small disasters. It improves for larger disasters because the number of single failures decreases.

TABLE VI: AE codes: Number of repair rounds.

Code	10%	20%	30%	40%	50%
AE(1,-,-)	6	7	9	10	10
AE(2,2,5)	3	6	9	17	30
AE(3,2,5)	3	4	7	10	15

4) Metric: Code Performance: At each round, our AE decoder computes 1 XOR between two available blocks for any data and parity blocks that is repaired. When data blocks cannot be repaired at the first round, the decode will do it at the second round if other required data or parity block becomes available. Fig. 13 shows that most data are repaired at the first round. The amount of blocks that is repaired per round decreases abruptly, normally the last round repairs only 1-2 blocks. The number of rounds that were needed to repair all data blocks are summarized in Table VI.

VI. Related Work

Data Entanglement in Censorship Resistant Systems. The main spirit is the creation of a legal conundrum with the hope that a censor will not delete documents if a vast number of legal documents must be deleted to permanently delete an illegal document. If the system creates strong dependencies among the files, data loss can be recovered by exploiting the relationships between files. The idea was explored in Tangler [25], Dagster [26], STEP-archival [41] and Entangled Cloud [42]. Aspnes et al. studied the theoretical aspects [23].
Data Entanglement to Reduce Storage. Friendstore [28] is a cooperative backup system. Its redundancy scheme is called xor(1,2). It provides simultaneously redundancy for files owned by different users. It only encodes data when disk space is limited. The novelty was to trade-off bandwidth for storage.
RAID. Disks arrays are organized in diverse ways: mirroring, RAID level 5 [1], [43], RAID level 6 [44], [45], two-dimensional RAID arrays [46], [47]. Single-entanglements are close to SSpiral codes [48], [49]. Row-Diagonal Parity, also known as RAID-DP, protects data against double disk failures [50]. Researchers explored RAID at a petabyte scale [51].
Other Relevant Codes. Parallel concatenated convolutional codes [52] and staircase codes [53] have some similarities with our work. Pyramid codes [54] combine local and global parities to increase read efficiency during failures. RESAR uses a virtual layer to survive double-failures with low storage overhead and efficient updates [8]. LDPC codes [55], [56] and regeneration codes [57] have valued properties for efficient repair but they are not widely understood by the system community and require complex parameterization [58], [59].

VII. Conclusion

Alpha entanglement codes are practical and flexible erasure codes designed to increase the reliability and integrity of a storage system, primarily for archiving data in unreliable environments. These mechanisms, based on the novel redundancy propagation, are not only very robust and sound, but they are also efficient and easily implementable in real systems. The encoder and decoder are lightweight—essentially based on exclusive-or operations—and offer promising trade-offs between security, resource usage and performance. We describe use cases for distributed and centralised storage systems.

The current research trend is to reduce to the maximum possible the storage overhead while improving the efficiency of repairs. While reducing storage costs satisfies industry needs, other problems are neglected. If we consider the potential savings on maintenance cost, AE codes make sense in a wide variety of scenarios whether they are decentralised or not.

Acknowledgement

The authors would like to thank the shepherd Yair Amir and the anonymous reviewers for their fruitful comments. V.E.G. thanks Jay Lofstead, Ahmed Amer, Kragen Sitaker, and Hugues Mercier for stimulating discussions, and to Luiz Barroso for answering questions regarding availability in global systems. This work was partially supported by SNSF Doc.Mobility 162014.

References

[1] D. A. Patterson, G. Gibson, and R. H. Katz, “A case for redundant arrays of inexpensive disks (RAID),” in *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*. ACM, 1988, pp. 109–116.

[2] J. S. Plank *et al.*, “A tutorial on reed-solomon coding for fault-tolerance in raid-like systems,” *Softw., Pract. Exper.*, vol. 27, no. 9, pp. 995–1012, 1997.

[3] R. Anderson *et al.*, “The eternity service,” in *Proceedings of PRAGOCRYPT*, vol. 96, 1996, pp. 242–252.

[4] J. J. Wylie, M. W. Bigrigg, J. D. Strunk, G. R. Ganger, H. Kiliccote, and P. K. Khosla, “Survivable information storage systems,” *Computer*, vol. 33, no. 8, pp. 61–68, 2000.

[5] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, “FARSITE: Federated, available, and reliable storage for an incompletely trusted environment,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 1–14, 2002.

[6] A. Haeberlen, A. Mislove, and P. Druschel, “Glacier: Highly durable, decentralized storage despite massive correlated failures,” in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association, 2005, pp. 143–158.

[7] B.-G. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, and R. Morris, “Efficient replica maintenance for distributed storage systems.” in *NSDI*, vol. 6, 2006, pp. 4–4.

[8] T. Schwarz, A. Amer, T. Kroeger, E. L. Miller, D. D. E. Long, and J.-F. Pâris, “RESAR: Reliable storage at exabyte scale,” in *Proceedings of the 24th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2016)*, Sep. 2016.

[9] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, “Availability in globally distributed storage systems.” in *OSDI*, 2010, pp. 61–74.

[10] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang *et al.*, “f4: Facebook’s warm blob storage system,” in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 383–398.

[11] D. S. Papailiopoulos and A. G. Dimakis, “Locally repairable codes,” *IEEE Transactions on Information Theory*, vol. 60, no. 10, pp. 5843–5855, 2014.

[12] O. Khan, R. C. Burns, J. S. Plank, W. Pierce, and C. Huang, “Rethinking erasure codes for cloud file systems: minimizing i/o for recovery and degraded reads.” in *FAST*, 2012, p. 20.

[13] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, “Erasure coding in windows azure storage,” in *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, 2012, pp. 15–26.

[14] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," in *Proceedings of the 39th international conference on Very Large Data Bases*, vol. 6, no. 5. VLDB Endowment, 2013.

[15] V. E. Galinanes and P. Felber, "Helical entanglement codes: An efficient approach for designing robust distributed storage systems," in *Symposium on Self-Stabilizing Systems*. Springer, 2013, pp. 32–44.

[16] V. Estrada-Galinanes, J.-F. Pâris, and P. Felber, "Simple data entanglement layouts with high reliability," in *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*.

[17] V. E. Galinanes and P. Felber, "Ensuring data durability with increasingly interdependent content," in *2015 IEEE International Conference on Cluster Computing*. IEEE, 2015, pp. 162–165.

[18] K. Greenan, E. L. Miller, and J. Wylie, "Reliability of xor-based erasure codes on heterogeneous devices," in *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2008)*, Jun. 2008, pp. 147–156.

[19] J. J. Wylie and R. Swaminathan, "Determining fault tolerance of xor-based erasure codes efficiently," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, 2007, pp. 206–215.

[20] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized erasure codes for distributed networked storage," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. SI, pp. 2809–2816, 2006.

[21] C. Gkantsidis and P. R. Rodriguez, "Network coding for large scale content distribution," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 4. IEEE, 2005, pp. 2235–2245.

[22] X. Zhang, G. Neglia, J. Kurose, and D. Towsley, "On the benefits of random linear coding for unicast applications in disruption tolerant networks," in *Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks, 2006 4th International Symposium on*. IEEE, 2006.

[23] J. Aspnes, J. Feigenbaum, A. Yampolskiy, and S. Zhong, "Towards a theory of data entanglement," *Theoretical Computer Science*, vol. 389, no. 1, pp. 26–43, 2007.

[24] D. N. Tran, F. Chiang, and J. Li, "Friendstore: cooperative online backup using trusted nodes," in *Proceedings of the 1st Workshop on Social Network Systems*. ACM, 2008, pp. 37–42.

[25] M. Waldman and D. Mazieres, "Tangler: a censorship-resistant publishing system based on document entanglements," in *Proceedings of the 8th ACM conference on Computer and Communications Security*. ACM, 2001, pp. 126–135.

[26] A. Stubblefield and D. S. Wallach, "Dagster: censorship-resistant publishing without replication," *Rice University, Dept. of Computer Science, Tech. Rep. TR01-380*, 2001.

[27] F. Oggier, A. Datta *et al.*, "Coding techniques for repairability in networked distributed storage systems," *Foundations and Trends in Communications and Information Theory*, vol. 9, no. 4, pp. 383–466, 2013.

[28] M. Lillibridge, S. Elnikety, A. Birrell, M. Burrows, and M. Isard, "A cooperative internet backup scheme," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '03. Berkeley, CA, USA: USENIX Association, 2003, pp. 3–3.

[29] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 4, pp. 337–350, 2010.

[30] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?" in *FAST*, vol. 7, 2007, pp. 1–16.

[31] D. Colarelli and D. Grunwald, "Massive arrays of idle disks for storage archives," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. IEEE Computer Society Press, 2002, pp. 1–11.

[32] B. Beach, "Backblaze open sources reed-solomon erasure coding source code," June 2015. [Online]. Available: https://www.backblaze.com/blog/reed-solomon/

[33] M. Holland and G. A. Gibson, *Parity declustering for continuous operation in redundant disk arrays*. ACM, 1992, vol. 27, no. 9.

[34] J. L. Hafner, V. Deenadhayalan, T. Kanungo, and K. Rao, "Performance metrics for erasure codes in storage systems," *IBM Res. Rep. RJ*, vol. 10321, 2004.

[35] C. Blake and R. Rodrigues, "High availability, scalable storage, dynamic peer networks: Pick two." in *HotOS*, vol. 3, 2003, p. 1.

[36] I. Adams, E. L. Miller, and D. S. Rosenthal, "Using storage class memory for archives with dawn, a durable array of wimpy nodes," Technical Report UCSC-SSRC-11-07, University of California, Santa Cruz, Tech. Rep., 2011.

[37] D. S. Rosenthal, D. C. Rosenthal, E. L. Miller, I. F. Adams, M. W. Storer, and E. Zadok, "The economics of long-term digital storage," *Memory of the World in the Digital Age, Vancouver, BC*, 2012.

[38] K. V. Vishwanath and N. Nagappan, "Characterizing cloud computing hardware reliability," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 193–204.

[39] V. Estrada-Galinanes, "A research playground for examining erasure codes," in *Proceedings of the 10th ACM International Systems and Storage Conference*. New York, NY, USA: ACM, 2017.

[40] W. Lin, D. M. Chiu, and Y. Lee, "Erasure code replication revisited," in *Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on*. IEEE, 2004, pp. 90–97.

[41] H. Mercier, M. Augier, and A. K. Lenstra, "Step-archival: Storage integrity and anti-tampering using data entanglement," in *2015 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2015.

[42] G. Ateniese, Ö. Dagdelen, I. Damgård, and D. Venturi, "Entangled cloud storage," *Future Generation Computer Systems*, vol. 62, pp. 104–118, 2016.

[43] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-performance, reliable secondary storage," *ACM Computing Surveys (CSUR)*, vol. 26, no. 2, pp. 145–185, 1994.

[44] T. J. Schwarz and W. Buckhard, "RAID organization and performance," in *[1992] Proceedings of the 12th International Conference on Distributed Computing Systems*. IEEE, 1992, pp. 318–325.

[45] W. A. Burkhard and J. Menon, "Disk array storage system reliability," in *Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers., The Twenty-Third International Symposium on*. IEEE, 1993, pp. 432–441.

[46] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, and D. A. Patterson, "Coding techniques for handling failures in large disk arrays," *Algorithmica*, vol. 12, no. 2, pp. 182–208, 1994.

[47] T. J. E. Schwarz, "Reliability and performance of disk arrays," Ph.D. dissertation, 1994.

[48] A. Amer, J.-F. Pâris, T. Schwarz, V. Ciotola, and J. Larkby-Lahet, "Outshining mirrors: Mttdl of fixed-order spiral layouts," in *Storage Network Architecture and Parallel I/Os, 2007. SNAPI. International Workshop on*. IEEE, 2007, pp. 11–16.

[49] A. Amer, D. D. Long, T. Schwarz, and J.-F. Paris, "Increased reliability with sspiral data layouts," in *2008 IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems*. IEEE, 2008, pp. 1–10.

[50] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, 2004, pp. 1–14.

[51] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, "Diskreduce: Raid for data-intensive scalable computing," in *Proceedings of the 4th Annual Workshop on Petascale Data Storage*. ACM, 2009, pp. 6–10.

[52] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Transactions on Communications*, vol. 44, no. 5, pp. 591–600, 1996.

[53] B. P. Smith, A. Farhood, A. Hunt, F. R. Kschischang, and J. Lodge, "Staircase codes: FEC for 100 gb/s OTN," *CoRR*, vol. abs/1201.4106, 2012. [Online]. Available: http://arxiv.org/abs/1201.4106

[54] C. Huang, M. Chen, and J. Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," *ACM Transactions on Storage (TOS)*, vol. 9, no. 1, p. 3, 2013.

[55] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[56] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman, and V. Stemann, "Practical loss-resilient codes," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. ACM, 1997, pp. 150–159.

[57] C. Suh and K. Ramchandran, "Exact regeneration codes for distributed storage repair using interference alignment," *arXiv preprint arXiv:1001.0107*, 2009.

[58] J. S. Plank and M. G. Thomason, "On the practical use of ldpc erasure codes for distributed storage applications," *Technical Report CS-03–510*, 2003.

[59] S. Jiekak, A.-M. Kermarrec, N. Le Scouarnec, G. Straub, and A. Van Kempen, "Regenerating codes: A system perspective," *ACM SIGOPS Operating Systems Review*, vol. 47, no. 2, pp. 23–32, 2013.