Noname manuscript No.
(will be inserted by the editor)

Self-repairing Codes

Local Repairability for Cheap & Fast Maintenance of Erasure Coded Data

Frédérique Oggier · Anwitaman Datta

Received: date / Accepted: date

Abstract Networked distributed data storage systems are essential to deal with the needs of storing massive volumes of data. Dependability of such a system relies on its fault tolerance (data should be available in case of node failures) as well as its maintainability (its ability to repair lost data to ensure redundancy replenishment over time). Erasure codes provide a storage efficient alternative to replication based redundancy in storage systems, ensuring the same fault tolerance at a lower storage overhead cost. Traditional erasure codes however have the drawback of entailing high communication overhead for maintenance, when encoded fragments are lost due to storage device failures, and need to be replenished in new nodes. We propose a new family of erasure codes called *self-repairing codes* (SRC) taking into account the peculiarities of distributed storage systems, specifically to improve its maintainability by 'localizing' the repairs. SRC have the property that encoded fragments can be repaired directly from other small subsets of (typically 2 or 3) encoded fragments. These code properties allow bandwidth efficient and fast recovery even in the presence of multiple failures, in turn translating into better system robustness. A concrete family of such *locally repairable codes*, namely, homomorphic self-repairing codes (HSRC) are proposed and various aspects and properties of the same are studied in detail and compared - quantitatively or

F. Oggier
School of Physical and Mathematical Sciences
Nanyang Technological University
Tel.: +65 6513 2026
Fax: +65 6515 9663
E-mail: frederique@ntu.edu.sg

A. Datta
School of Computer Engineering
Nanyang Technological University
Tel.: +65 6790 4855
Fax: +65 6792 6559
E-mail: anwitaman@ntu.edu.sg

qualitatively (as may be suitable) with respect to other codes including traditional erasure codes as well as some recent representative codes designed specifically for storage applications.

Keywords Erasure codes · Local Repair · Networked Storage

1 Introduction

Various genres of networked storage systems, such as decentralized peer-to-peer storage systems or dedicated infrastructure based data-centers and storage area networks, have gained prominence in recent years. Because of storage node failures, or user attrition in a peer-to-peer system, redundancy is essential for fault tolerance in networked storage systems. This redundancy can be achieved using either replication (replication of the same data within one system, or using backups or remote mirroring [9]), or (erasure) coding techniques, or a mix of the two. Erasure codes require an object to be split into k parts, and mapped into n encoded fragments, such that any k encoded fragments are adequate to reconstruct the original object. Such coding techniques play a prominent role in providing storage efficient redundancy, and are particularly effective for storing large data objects and for archival and data back-up applications (e.g., CleverSafe [3], Wuala [13]).

Redundancy is lost over time because of various reasons such as node failures or attrition, and mechanisms to replenish redundancy are essential to the system maintainability. While erasure codes are efficient in terms of storage overhead, maintenance of lost redundancy entail huge overheads [25]. A naive approach to replace a single missing fragment will require that k encoded fragments are first fetched in order to create the original object, from which the missing fragment is recreated and replenished. This essentially means that for every lost fragment, k-fold more network traffic is incurred.

Engineering solutions can partly mitigate the high maintenance overheads. A 'hybrid' strategy, where a full replica of the object is additionally maintained [25], ensures that the amount of network traffic equals the amount of lost data.[1] A spate of recent works [4,6] argue that the hybrid strategy adds storage inefficiency and system complexity, besides being a bottleneck. Another possibility is to apply lazy maintenance [2,5], whereby maintenance is delayed in order to amortize the maintenance of several missing fragments. Lazy strategies additionally avoid maintenance due to temporary failures. Procrastinating repairs however may lead to a situation where the system becomes vulnerable, e.g. to bursty/correlated failures, and thus may require a much larger amount of redundancy to start with. Furthermore, the maintenance operations may lead to spikes in network resource usage [11].

This motivates the question of the dependability of erasure coding based distributed storage systems, keeping the reliability that erasure coding pro-

[1] In this paper, we use the terms 'fragment' and 'block' interchangeably. Depending on the context, the term 'data' is used to mean either fragment(s) or object(s).

vides, while looking for better solutions to improve the maintainability. Network storage has benefitted from the research done in coding for communication channels by using erasure codes as black boxes that provide efficient distribution and reconstruction of the stored objects. It however involves different challenges but also opportunities not addressed by classical erasure codes. Recently, there has thus been a renewed interest [4,6,7,21,10,26,8,23,29] in designing codes that are optimized to deal with the vagaries of networked storage, particularly focusing on the maintainability issue, more precisely, on mechanisms that allow new nodes to be provided with fragments of stored data to compensate for the departure of nodes from the system, and replenish the level of redundancy (in order to tolerate further faults in future). Security in erasure coding based distributed storage systems is another crucial aspect, see e.g. [1], but it is out of the scope of our current study in this paper. How the local repairability property of the proposed codes may affect the security aspects would indeed be an interesting new line of work for future. The goal of this paper is to propose a new family of codes called *self-repairing codes* (SRC), which are tailored to fit well the maintainability of typical networked storage environments.

As any linear (n, k, d) erasure code over a q-ary alphabet, a SRC is formally a linear map $c : \mathbb{F}_{q^k} \rightarrow \mathbb{F}_{q^n},\ \mathbf{s} \mapsto c(\mathbf{s})$ which maps a k-dimensional vector $\mathbf{s}$ to an n-dimensional vector $c(\mathbf{s})$. The set C of codewords $c(\mathbf{s})$, $\mathbf{s} \in \mathbb{F}_{q^k}$, forms the code (or codebook). The quantity k/n is called the rate of the code. The third parameter d refers to the minimum distance of the code: $d = \min_{\mathbf{x} \neq \mathbf{y} \in C} d(\mathbf{x}, \mathbf{y})$ where the Hamming distance $d(\mathbf{x}, \mathbf{y})$ counts the number of positions at which the coefficients of $\mathbf{x}$ and $\mathbf{y}$ differ. The minimum distance describes how many erasures can be tolerated, which is known to be at most $n - k$, achieved by maximum distance separable (MDS) codes. MDS codes thus allow to recover any codeword out of k coefficients. Though SRCs are not MDS codes, their definition mimics the MDS property in terms of repair: we define *self-repairing codes* as (n, k) codes that encode k fragments of an object into n encoded fragments to be stored at n nodes, with the properties that: (a) *encoded fragments can be repaired directly from other subsets of encoded fragments by downloading less data than the size of the complete object.* Based on the analogy with the error correction capability of erasure codes, which is of any $n-k$ losses independently of which ones, (b) *a fragment can be repaired from a small fixed number ($< k$) of encoded fragments, depending only on how many encoded blocks are missing, independently of which specific blocks are missing.* The term "self-repair" was chosen to emphasize the property of the codes which enable repairs using a very small subset of the encoded pieces, particularly, smaller in amount than the size of the complete data.

Since our early works on SRCs in 2010 [18,19], research on codes with localized repairability (where the number of nodes contacted to carry out a repair is less than k) has gained traction, and inspired by the term 'locally decodable codes', have come to be known as 'locally repairable codes' (this term was coined in [20] in 2012). The idea of localized repairability precedes our work and was already present in [16], where the notion of local/global

parities was introduced to deal with degraded reads. The idea that SRCs added was the ability to carry out local repairs for all the encoded pieces, and not only a specific subset. Broadly speaking, two flavors of research about LRC have emerged over time, one looking at the code instances themselves [20,28], or [14], which proposes a hybrid scheme mixing ideas from LRCs and regenerating codes [21], and the other, looking theoretically at the fundamental trade-offs involved in achieving local repairability vis-a-vis code rate or fault-tolerance [12,15,17]. In this work, our approach is to pursue the former, i.e., design of the codes themselves, and analyze their performance. However, a recent independent work [17] defines 'optimality' with respect to the repair bandwidth v.s. the code's storage overhead, and shows that the codes [18] we propose are indeed optimal.

In order to achieve local repairability, SRCs (and LRCs in general) naturally require more storage overhead than MDS erasure codes for equivalent fault tolerance (static resilience). We will see more precisely later on that there is a tradeoff between the ability to self-repair and this extra storage overhead: SRC could be tuned to be MDS at the price of losing the self-repair property, and conversely, the facility to self-repair can be adapted based on the amount of extra redundancy introduced. Consequently, SRCs can recreate the whole object with k fragments, though unlike for erasure codes, these are not arbitrary k fragments, however, many such k combinations can be found (see Section 4 for more details).

Note that even for traditional erasure codes, the property (a) may coincidentally be satisfied, but in absence of a systematic mechanism this serendipity cannot be leveraged. In that respect, hierarchical codes (HCs) [6] may be viewed as a way to do so, and are thus the closest example of construction we have found in the literature, though they do not give any guarantee on the number of blocks needed to repair given the number of losses, i.e., property (b) is not satisfied, and has no deterministic guarantee for achieving property (a) either. We may say at a very high level that SRC design features mitigate the drawbacks of HCs.

While motivated by the same problem as regenerating codes (RGC) and HCs, that of efficient maintenance of lost redundancy in coding based distributed storage systems, the approach of self-repairing codes (SRC) tries to do so at a somewhat different point of the design space. We try to minimize the number of nodes necessary to reduce the reconstruction of a missing block, which automatically translates into lower bandwidth consumption ([17] shows independently that our codes are in fact optimal w.r.to repair cost given a specific storage overhead), but also lower computational complexity of maintenance, as well as the possibility for faster and parallel replenishment of lost redundancy. Thus SRCs allow *light weight* (in terms of communication and computation overhead) and *flexible* (in terms of flexibility in the number of options to carry out specific repairs, which in turn allow parallel and fast repairs), that is, *agile* maintenance, of networked storage systems.

In this work, we make the following *contributions*:
(i) We propose a family of codes, self-repairing codes (SRC), designed specif-

ically as an alternative to erasure codes (EC) for providing redundancy in networked storage systems, which minimizes the number d of nodes contacted during repair, in particular, we consider the case $d < k$.
(ii) We treat the design of non-MDS storage codes: like ECs, SRCs allow recovery of the whole object using k encoded fragments, but unlike in ECs, these are not any arbitrary k fragments, though numerous suitable combinations exist.
(iii) We provide a deterministic code construction called *Homomorphic Self-Repairing Code* (HSRC), that generalizes the HSRCs introduced in [18] by extending the base field, thus allowing more flexibility in the code design.
(iv) HSRC self-repair operations are computationally efficient. It is done by XORing encoded blocks, each of them containing information about all fragments of the object. The encoding itself is however done through polynomial evaluation (similar to popular ECs).
(v) We show that for an equivalent static resilience, marginally more storage is needed than traditional ECs to achieve the self-repairing property (despite sacrificing the MDS property), conciliating fault-tolerance and maintainability. This generalizes the analysis of [18].
(vi) The need of few blocks to reconstruct a lost block naturally translates to low overall bandwidth consumption for repair operations. SRCs allow for both eager as well as lazy repair strategies for equivalent overall bandwidth consumption for a wide range of practical system parameter choices. They also outperform lazy repair with the use of traditional erasure codes for many practical parameter choices.
(vii) We show that by allowing parallel and independent repair of different encoded blocks (leveraging on the low value of d), SRCs facilitate fast replenishment of lost redundancy, allowing a much quicker system recovery from a vulnerable state than is possible with traditional codes. This also implies a distribution of the repair related tasks across different nodes, thus avoiding bottlenecks or overloading any specific node or part of the network.

2 Related work

In [4], Dimakis et al. propose regenerating codes (RGC) by using classical erasure codes as a black box over a network which implements random linear network coding and propose leveraging the properties of network coding to improve the maintenance of the stored data. Network information flow based analysis shows the possibility to replace a missing fragment using network traffic equalling the volume of lost data, however only by communicating with all the $n - 1$ remaining blocks. Consequently, to the best of our knowledge, regenerating codes literature generally does not discuss how it compares with engineering solutions like lazy repair, which amortizes the repair cost by initiating repairs only when several fragments are lost. Furthermore, for RGCs to work, even sub-optimally, it is essential to communicate with at least k other nodes to reconstruct any missing fragment. Thus, while the volume of data-

transfer for maintenance is lowered, RGCs are expected to have higher protocol overheads, implementation and computational complexity. For instance, it is noted in [7] that a randomized linear coding based realization of RGCs takes an order of magnitude more computation time than standard erasure codes for both encoding and decoding. Explicit codes constructions of RGCs are given in [21,22]. Recently, collaborative RGC were introduced independently [10, 26], where it was shown that collaboration among new nodes joining the network and participating to the repair process can improve on traditional RGC, in terms of both (i) storage-bandwidth trade-off, that is the amount of data that is stored at each node with respect to that which is downloaded by new nodes during repair, and (ii) number of simultaneous failures tolerated. While such analysis determines constraint on achievability, for classical as well as collaborative RGC, code constructions for collaborative RGC are even sparser [27].

In [6], the authors make the simple observation that encoding two bits into three by XORing the two information bits has the property that any two encoded bits can be used to recover the third one. They then propose an iterative construction where, starting from small erasure codes, a bigger code, called hierarchical code (HC), is built by XORing subblocks made by erasure codes or combinations of them. Thus a subset of encoded blocks is typically enough to regenerate a missing one. However, the size of this subset can vary, from the minimal to the maximal number of encoded subblocks, determined by not only the number of lost blocks, but also the specific lost blocks. So given some lost encoded blocks, this strategy may need an arbitrary number of other encoded blocks to repair. Pyramid codes [16] explore similar ideas.

Following our first works [18,19] from 2010-2011, there has been a tremendous interest [17,20,28,12,15] both in designing codes with local repairability, and determining the feasibility of specific code parameters. With respect to these follow-up works, the main distinctions of the presented works include the simultaneous achievement of the following desirable properties — code optimality (proven independently in [17]), repair locality of both data as well as parity blocks, the availability of multiple choices to carry out individual repairs; and the novelties of the study includes a rigorous analysis of the fault-tolerance (static resilience) of the proposed codes which contains the static resilience analysis of [18] as a particular case, and exploration of other desirable system properties such as the parallelization of repairs to achieve fast recovery from multiple simultaneous failures.

3 Homomorphic Self-Repairing Codes

In what follows, we denote finite fields by $\mathbb{F}$, and finite fields without the zero element by $\mathbb{F}^*$. The cardinality of $\mathbb{F}$ is given by its index, that is, $\mathbb{F}_2$ is the binary field with two elements, which is nothing else than the two bits 0 and 1, with addition and multiplication modulo 2, $\mathbb{F}_q$ is the finite field with q elements, and $\mathbb{F}_Q$ is the finite field with Q elements. If $q = 2^t$, $Q = q^m = 2^{tm}$,

$\mathbb{F}_4$	$\mathbb{F}_8$	$\mathbb{F}_{16}$	
0	0	0	$w^7 = w^3 + w + 1$
1	1	1	$w^8 = w^2 + 1$
w	w	w	$w^9 = w^3 + w$
w^2	w^2	w^2	$w^{10} = w^2 + w + 1$
=	$w^3 = w + 1$	w^3	$w^{11} = w^3 + w^2 + w$
$w + 1$	$w^4 = w^2 + w$	$w^4 = w + 1$	$w^{12} = w^3 + w^2 + w + 1$
	$w^5 = 1 + w^2 + w$	$w^5 = w^2 + w$	$w^{13} = w^3 + w^2 + 1$
	$w^6 = w^2 + 1$	$w^6 = w^3 + w^2$	$w^{14} = w^3 + 1$

Table 1 The finite fields $\mathbb{F}_4, \mathbb{F}_8$ and $\mathbb{F}_{16}$, where w denotes the generator of their respective multiplicative groups $\mathbb{F}_4^*$, $\mathbb{F}_8^*$ and $\mathbb{F}_{16}^*$.

for some positive integers m and t, an element $\mathbf{x} \in \mathbb{F}_Q$ can be represented by an m-dimensional vector $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_m)$ where $\mathbf{x}_i \in \mathbb{F}_q$, $i = 1, \ldots, m$, by fixing an $\mathbb{F}_q$-basis of $\mathbb{F}_Q$. Similarly, each coefficient $\mathbf{x}_i$ can be written as $\mathbf{x}_i = (x_{i1}, \ldots, x_{it})$, $x_{ij} \in \mathbb{F}_2$, so that $\mathbf{x}$ may alternatively be seen as a tm-dimensional binary vector $\mathbf{x} = (x_{11}, \ldots, x_{1t}, \ldots, x_{m1}, \ldots, x_{mt})$. We say that $\mathbf{x}$ is a vector of size m to refer to m coefficients in $\mathbb{F}_q$, so that q determines the unit in which the size m is measured: for example, if $q = 2$, $\mathbf{x}$ is m bit long. To do explicit computations in the finite field $\mathbb{F}_q$, it is often convenient to use a fixed generator of the multiplicative group $\mathbb{F}_q^*$, that we will denote by w. A generator has the property that $w^{q-1} = 1$, and there is no smaller positive power of w for which this is true. Examples of finite fields that will be used later on are given in Table 1.

3.1 Encoding

Let $\mathbf{o}$ be an object of size M to be stored over a network of n nodes, that is $\mathbf{o} \in \mathbb{F}_{q^M}$, and let k be a positive integer such that k divides M. We can write

$$\mathbf{o} = (\mathbf{o}_1, \ldots, \mathbf{o}_k), \ \mathbf{o}_i \in \mathbb{F}_{q^{M/k}}$$

which requires the use of a (n, k) code over $\mathbb{F}_{q^{M/k}}$, that maps $\mathbf{o}$ to an Mn/k-dimensional vector $\mathbf{x}$, or equivalently, an n-dimensional vector

$$\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n), \ \mathbf{x}_i \in \mathbb{F}_{q^{M/k}},$$

after which each $\mathbf{x}_i$ is given to a node to be stored. The theory developed below assumes this model, which can be adjusted to fit real scenarios as elaborated later in Example 5.

Since the work of Reed and Solomon [24], it is known that linear coding can be done via polynomial evaluation. In short, take an object $\mathbf{o} = (\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_k)$ of size M, with each $\mathbf{o}_i$ in $\mathbb{F}_{q^{M/k}}$, and create the polynomial

$$p(X) = \mathbf{o}_1 + \mathbf{o}_2 X + \ldots \mathbf{o}_k X^{k-1} \in \mathbb{F}_{q^{M/k}}[X].$$

Now evaluate $p(X)$ in n elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_{q^{M/k}}^*$, to get the codeword

$$(p(\alpha_1), \ldots, p(\alpha_n)), \ k < n \leq q^{M/k} - 1.$$

Example 1 Suppose that $q = 2$, so that the size of the object is measured in bits. Take the 4 bit long object $\mathbf{o} = (o_1, o_2, o_3, o_4)$, and create $k = 2$ fragments: $\mathbf{o}_1 = (o_1, o_2) \in \mathbb{F}_4$, $\mathbf{o}_2 = (o_3, o_4) \in \mathbb{F}_4$. We use a $(3, 2)$ Reed-Solomon code over $\mathbb{F}_4$, to store the file in 3 nodes. Recall that $\mathbb{F}_4 = \{(a_0, a_1),\ a_0, a_1 \in \mathbb{F}_2\} = \{a_0 + a_1 w,\ a_0, a_1 \in \mathbb{F}_2\}$ where $w^2 = w+1$. Thus we can alternatively represent each fragment as: $\mathbf{o}_1 = o_1 + o_2 w \in \mathbb{F}_4$, $\mathbf{o}_2 = o_3 + o_4 w \in \mathbb{F}_4$. The encoding is done by first mapping the two fragments into a polynomial $p(X) \in \mathbb{F}_4[X]$:

$$p(X) = (o_1 + o_2 w) + (o_3 + o_4 w)X,$$

and then evaluating $p(X)$ into the three non-zero elements of $\mathbb{F}_4$, to get a codeword of length 3:

$$(p(1), p(w), p(w + 1))$$

where $p(1) = o_1 + o_3 + w(o_2 + o_4)$, $p(w) = o_1 + o_4 + w(o_2 + o_3 + o_4)$, $p(w^2) = o_1 + o_3 + o_4 + w(o_2 + o_3)$, so that each node gets two bits to store: $(o_1 + o_3, o_2 + o_4)$ at node 1, $(o_1 + o_4, o_2 + o_3 + o_4)$ at node 2, $(o_1 + o_3 + o_4, o_2 + o_3)$ at node 3.

Definition 1 *We call homomorphic self-repairing code, denoted by* $\mathrm{HSRC}(n, k)$*, the code obtained by evaluating the polynomial*

$$p(X) = \sum_{i=0}^{k-1} p_i X^{q^i} \in \mathbb{F}_{q^{M/k}}[X] \tag{1}$$

in n non-zero values $\alpha_1, \ldots, \alpha_n$ of $\mathbb{F}_{q^{M/k}}$ to get an n-dimensional codeword

$$(p(\alpha_1), \ldots, p(\alpha_n)),$$

where $p_i = \mathbf{o}_{i+1}$, $i = 0, \ldots, k-1$ and each $p(\alpha_i)$ is given to node i for storage.

This definition translates into a general methodology to construct storage codes, similarly as for Reed-Solomon codes, since the coefficients of the polynomial correspond to the object to be stored, and the choice of the points $\alpha_1, \ldots, \alpha_n$ (including the choice of n) defines code parameters. The consequences of the choice of n, k, and $\alpha_1, \ldots, \alpha_n$ are discussed in Subsection 3.2 below.

In particular, we need the code parameters (n, k) to satisfy

$$k < n \leq q^{M/k} - 1. \tag{2}$$

The analysis that follows refers to this family of self-repairing codes.

3.2 Self-repair

Since we work over finite fields that contain $\mathbb{F}_2$, recall that all operations are done in characteristic 2, that is, additions are performed modulo 2. Let $a, b \in \mathbb{F}_{q^m}$, for some $m \geq 1$ and $q = 2^t$. Then we have that $(a + b)^2 = a^2 + 2ab + b^2 = a^2 + b^2$ since $2ab \equiv 0 \mod 2$, and consequently one can iteratively compute

$$(a + b)^{2^i} = [(a + b)^2]^{2^{i-1}} = [a^2 + b^2]^{2^{i-1}} = \ldots = a^{2^i} + b^{2^i},\ i \geq 1. \tag{3}$$

Definition 2 *A linearized polynomial*[2] $p(X)$ *over* $\mathbb{F}_Q$, $Q = q^m$, *has the form*

$$p(X) = \sum_{i=0}^{k-1} p_i X^{Q^i}, \ p_i \in \mathbb{F}_Q.$$

More generally, one can consider a polynomial $p(X)$ over $\mathbb{F}_Q$, $Q = q^m$, of the form:

$$p(X) = \sum_{i=0}^{k-1} p_i X^{s^i}, \ p_i \in \mathbb{F}_Q,$$

where $s = q^l$, $1 \leq l \leq m$ ($l = m$ makes $p(X)$ a linearized polynomial). These polynomials share the following useful property:

Lemma 1 *Let* $a, b \in \mathbb{F}_{q^m}$ *and let* $p(X)$ *be the polynomial given by* $p(X) = \sum_{i=0}^{k-1} p_i X^{s^i}$, $s = q^l$, $m \geq l \geq 1$. *We have*

$$p(ua + vb) = up(a) + vp(b), \ u, v \in \mathbb{F}_s.$$

Proof If we evaluate $p(X)$ in $ua + vb$, we get

$$p(ua + vb) = \sum_{i=0}^{k-1} p_i (ua + vb)^{s^i} = \sum_{i=0}^{k-1} p_i ((ua)^{s^i} + (vb)^{s^i})$$

by (3), and

$$p(ua + vb) = \sum_{i=0}^{k-1} p_i (ua^{s^i} + vb^{s^i}) = u \sum_{i=0}^{k-1} p_i a^{s^i} + v \sum_{i=0}^{k-1} p_i b^{s^i}$$

using the property that $u^s = u$ for $u \in \mathbb{F}_s$.

We now define a weakly linearized polynomial as

Definition 3 *A weakly linearized polynomial* $p(X)$ *over* $\mathbb{F}_Q$, $Q = q^m$, *has the form*

$$p(X) = \sum_{i=0}^{k-1} p_i X^{q^i}, \ p_i \in \mathbb{F}_Q.$$

We chose the name weakly linearized polynomial, since we only retain the $\mathbb{F}_q$-linearity, namely:

Corollary 1 *Let* $a, b \in \mathbb{F}_{q^m}$ *and let* $p(X)$ *be a weakly linearized polynomial given by* $p(X) = \sum_{i=0}^{k-1} p_i X^{q^i}$. *We have*

$$p(ua + vb) = up(a) + vp(b), \ u, v \in \mathbb{F}_q. \tag{4}$$

In particular

$$p(a + b) = p(a) + p(b). \tag{5}$$

[2] Linearized polynomials are also called additive polynomials.

It is the choice of a weakly linearized polynomial in (1) that enables self-repair.

Example 2 Consider the polynomial

$$p(X) = p_0X + p_1X^2 + p_2X^4 \in \mathbb{F}_8[X].$$

We have (see Table 1 for $\mathbb{F}_8$ arithmetic)

$$p(w+1) = p_0(w+1) + p_1(w^2+1) + p_2(w^4+1) = p(w) + p(1)$$

however

$$p(w^2) \neq p(w)^2$$

since $p_i^2 = p_i$ if and only if $p_i \in \mathbb{F}_2$.

A codeword from HSRC(n,k) is then of the form $(p(\alpha_1), \ldots, p(\alpha_n))$, where $p(X)$ is a weakly linearized polynomial. Since $\mathbb{F}_{q^{M/k}}$ contains a $\mathbb{F}_q$-basis $B = \{b_1, \ldots, b_{M/k}\}$, the α_i, $i = 1, \ldots, n$, can be expressed as $\mathbb{F}_q$-linear combinations of the basis elements, and we have from Lemma 1 that

$$\alpha_i = \sum_{j=1}^{M/k} \alpha_{ij} b_j,\ \alpha_{ij} \in \mathbb{F}_q \Rightarrow p(\alpha_i) = \sum_{j=1}^{M/k} \alpha_{ij} p(b_j).$$

In words, that means that if p is evaluated in the elements of the basis B ($p(b_1), \ldots, p(b_{M/k})$ are computed), or any other basis, then any encoded fragment $p(\alpha_i)$ can be obtained as an $\mathbb{F}_q$-linear combination of other encoded fragments.

Controlling the amount of redundancy. The amount of redundancy allowing self-repair introduced in the coding scheme can be controlled through two mechanisms:

1) Firstly, given k fragments, there are different values of n, and different choices of $\{\alpha_1, \ldots, \alpha_n\}$ that can be chosen to define a self-repairing code. Let us denote by n_{max} the maximum value that n can take, namely $n_{max} = q^{M/k} - 1$. By choosing the set of α_i to form a subspace of $\mathbb{F}_{n_{max}+1}$, we can reduce the redundancy while maintaining a particularly nice symmetric structure of the code. In the extreme case where $\alpha_1, \ldots, \alpha_n$ are contained in B, the code has no self-repairing property, and is in fact a MDS code. Thus SRC can be tuned to provide the desired amount of redundancy, from MDS and no self-repair, to the maximal amount of self-repair with n_{max}.

2) As seen in Lemma 1, the power s of X^{s^i} in the weakly linearized polynomial $p(X)$ determines the $\mathbb{F}_s$-linearity of $p(X)$. Consequently, the bigger s, the more redundancy since from (4)

$$p(ua + vb) = up(a) + vp(b),\ u, v \in \mathbb{F}_q,$$

meaning that the encoded fragment $p(ua+vb)$ can be repaired by contacting two nodes $p(a), p(b)$ in as many ways as there are ways for writing $ua + vb$, namely $(q-1)^2$:

$$ua + vb = u(u')^{-1}u'a + v(v')^{-1}v'b,\ u', v' \neq 0,\ u', v' \in \mathbb{F}_q.$$

In the particular case where $s = 2$, we obtain from (5) an XOR-like structure

$$p(a + b) = p(a) + p(b).$$

However, it is worth remarking that though the encoded fragments can be thus obtained as XORs of each other, each fragment is actually containing information about all the different fragments, which is very different than a simple XOR of the data itself. In particular, HSRC is not a systematic code. The implications of lack of systematic property will be discussed in Subsection 6.2.

Computational complexity of self-repair. In terms of computational complexity, the case $s = 2$ implies that the cost of a block reconstruction is that of some XORs (one in the most favorable case, when two terms are enough to reconstruct a block, up to $k-1$ in the worst case), independently of q, since if $q = 2^t$, the addition in $\mathbb{F}_q$ is done by addition modulo 2 componentwise. The cost increases if one would like to exploit the $\mathbb{F}_q$-linearity. Indeed, repairing through

$$p(ua + vb) = up(a) + vp(b),\ u, v \in \mathbb{F}_q,$$

further requires two multiplications in $\mathbb{F}_q$.

3.3 Decoding

That decoding is possible is guaranteed by either Lagrange interpolation, or by considering a system of linear equations, assuming that

$$k \leq M/k, \tag{6}$$

as detailed below.

Lagrange interpolation. Given k fragments $p(\alpha_{i_1}), \ldots, p(\alpha_{i_k})$ such that $\alpha_{i_1}, \ldots, \alpha_{i_k}$ are $\mathbb{F}_q$-linearly independent, the node that wants to reconstruct the file computes $q^k - 1$ linear combinations of the k fragments, which gives, thanks to the homomorphic property (4), $q^k - 1$ points in which p is evaluated. Lagrange interpolation guarantees that it is enough to have $q^{k-1} + 1$ points (which we have, since $q^k - 1 \geq q^{k-1} + 1$ for $k \geq 2$) to reconstruct uniquely the polynomial p and thus the object. This requires

$$q^{k-1} + 1 \leq q^{M/k} - 1,$$

namely there must be enough points in which to evaluate the polynomial, which holds subject to (6):

$$q^k \leq q^{M/k} \Rightarrow q^{k-1} + 1 \leq q^k - 1 \leq q^{M/k} - 1.$$

Solving a system of linear equations. Alternatively, one can consider decoding as solving a system of linear equations. Given k $\mathbb{F}_q$-linearly independent fragments, say $p(\alpha_{i_1}), \ldots, p(\alpha_{i_k})$, we can write

$$\begin{pmatrix} \alpha_{i_1} & \alpha_{i_1}^q & \alpha_{i_1}^{q^2} & \ldots & \alpha_{i_1}^{q^{k-1}} \\ \alpha_{i_2} & \alpha_{i_2}^q & \alpha_{i_2}^{q^2} & \ldots & \alpha_{i_2}^{q^{k-1}} \\ \vdots & & \vdots & & \\ \alpha_{i_k} & \alpha_{i_k}^q & \alpha_{i_k}^{q^2} & \ldots & \alpha_{i_k}^{q^{k-1}} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{k-1} \end{pmatrix} = \begin{pmatrix} p(\alpha_{i_1}) \\ p(\alpha_{i_2}) \\ \vdots \\ p(\alpha_{i_k}) \end{pmatrix},$$

and the problem of recovering the object reduces to solving the above system of linear equations. Note that since $\mathbb{F}_{q^{M/k}}$ is a vector space of dimension M/k over $\mathbb{F}_q$, condition (6) is needed to guarantee that there exist k $\mathbb{F}_q$-linearly independent fragments.

3.4 Worked out examples

Let us first illustrate the choices of the code parameters (n, k), before detailing some code constructions.

We recall that the parameters (n, k) of an HSRC(n, k) code must satisfy conditions (2) and (6):

$$k < n \leq q^{M/k} - 1,\ k \leq M/k.$$

Thus for any choice of k: (1) pick any M which is a multiple of k (zero padding can be used to remove the constraint on the real size of the object), (2) define $n_{max} = q^{M/k} - 1$, (3) pick any n such that $n \geq q^k - 1,\ n < n_{max}$ which is a power of q minus 1 (this last condition is not completely necessary but ensures symmetry as already mentioned above).

Some examples of small parameters (n, k) for $q = 2$ are (1) $k = 2$, $M = 6$, $n_{max} = 7$, $n = 3$, (2) $k = 2$, $M = 8$, $n_{max} = 15$, $n = 3$ or $n = 7$, (3) $k = 3$, $M = 9$, $n_{max} = 7$, (4) $k = 3$, $M = 12$, $n_{max} = 15$, $n = 7$.

Example 3 Take a data file $\mathbf{o} = (o_1, \ldots, o_{12})$ of $M = 12$ bits ($q = 2$), and choose $k = 3$ fragments. We have that $M/k = 4$, which satisfies (6), that is $k = 3 \leq M/k = 4$.

The file $\mathbf{o}$ is cut into 3 fragments $\mathbf{o}_1 = (o_1, \ldots, o_4)$, $\mathbf{o}_2 = (o_5, \ldots, o_8)$, $\mathbf{o}_3 = (o_9, \ldots, o_{12}) \in \mathbb{F}_{2^4}$. Let w be a generator of the multiplicative group $\mathbb{F}_{2^4}^*$, such that $w^4 = w + 1$. The polynomial used for the encoding is

$$p(X) = \sum_{i=1}^{4} o_i w^{i-1} X + \sum_{i=1}^{4} o_{i+4} w^{i-1} X^2 + \sum_{i=1}^{4} o_{i+8} w^{i-1} X^4.$$

The n-dimensional codeword is obtained by evaluating $p(X)$ in n elements of $\mathbb{F}_{2^4}$, $n \leq 15 = n_{max}$ by (2).

missing fragment(s)	pairs to reconstruct missing fragment(s)
$p(1)$	$(p(w), p(w^4))$;$(p(w^2), p(w^8))$;$(p(w^5), p(w^{10}))$
$p(w)$	$(p(1), p(w^4))$;$(p(w^2), p(w^5))$;$(p(w^8), p(w^{10}))$
$p(w^2)$	$(p(1), p(w^8))$;$(p(w), p(w^5))$;$(p(w^4), p(w^{10}))$
$p(1)$ and $p(w)$	$(p(w^2), p(w^8))$ or $(p(w^5), p(w^{10}))$ for $p(1)$ $(p(w^8), p(w^{10}))$ or $(p(w^2), p(w^5))$ for $p(w)$
$p(1)$ and $p(w)$ and $p(w^2)$	$(p(w^5), p(w^{10}))$ for $p(1)$ $(p(w^8), p(w^{10}))$ for $p(w)$ $(p(w^4), p(w^{10}))$ for $p(w^2)$

Table 2 Ways of reconstructing missing fragment(s) in Example 3

For $n = 4$, if we evaluate $p(X)$ in w^i, $i = 0, 1, 2, 3$, then the 4 encoded fragments $p(1), p(w), p(w^2), p(w^3)$ are $\mathbb{F}_2$-linearly independent and there is no self-repair possible.

Now for $n = 7$, and say, $1, w, w^2, w^4, w^5, w^8, w^{10}$, we get:

$$(p(1), p(w), p(w^2), p(w^4), p(w^5), p(w^8), p(w^{10})).$$

Suppose node 5 which stores $p(w^5)$ goes offline. A new comer can get $p(w^5)$ by asking for $p(w^2)$ and $p(w)$, since

$$p(w^5) = p(w^2 + w) = p(w^2) + p(w).$$

Table 2 shows other examples of missing fragments and which pairs can reconstruct them, depending on if 1, 2, or 3 fragments are missing at the same time.

As for decoding, since $p(X)$ is of degree 4, a node that wants to recover the data needs $k = 3$ $\mathbb{F}_2$-linearly independent fragments, say $p(w), p(w^2), p(w^3)$, out of which it can generate $p(aw + bw^2 + cw^3)$, $a, b, c \in \{0, 1\}$. Out of the 7 non-zero coefficients, 5 of them are enough to recover p. Finally, if the rate of this code is $3/7 \simeq 0.43$.

Example 4 Take now a data file $\mathbf{o} = (o_1, \ldots, o_{16})$ with $M = 16$ and $q = 8$, and choose $k = 4$ fragments. We have that $M/k = 4$, which satisfies (6), that is $k \leq M/k$.

The file $\mathbf{o}$ is cut into 4 fragments $\mathbf{o}_1 = (o_1, \ldots, o_4)$, $\mathbf{o}_2 = (o_5, \ldots, o_8)$, $\mathbf{o}_3 = (o_9, \ldots, o_{12})$, $\mathbf{o}_4 = (o_{13}, \ldots, o_{16}) \in \mathbb{F}_{8^4}$. Let w be a generator of the multiplicative group of $\mathbb{F}_8$, and ν be a generator of the multiplicative group of $\mathbb{F}_{8^4}/\mathbb{F}_8$ such that $\nu^4 = \nu^3 + w$. The polynomial used for the encoding can be either

$$p(X) = \sum_{j=0}^{3} \sum_{i=0}^{3} o_{i+1+4j} \nu^i X^{8^j}$$

or

$$p'(X) = \sum_{j=0}^{3} \sum_{i=0}^{3} o_{i+1+4j} \nu^i X^{2^j}.$$

The n-dimensional codeword is obtained by evaluating $p(X)$ in n elements of $\mathbb{F}_{8^4}$, $n \leq 8^4 - 1 = n_{max}$ by (2).

Now for $n = 63$, and say, $\{u + \nu v,\ u, v \in \mathbb{F}_8,\ (u, v) \neq (0, 0)\}$, we get:

$$\{p(u + \nu v) :\ u, v \in \mathbb{F}_8,\ (u, v) \neq (0, 0)\},$$

respectively

$$\{p'(u + \nu v) :\ u, v \in \mathbb{F}_8,\ (u, v) \neq (0, 0)\}.$$

Let us give an example of repair in both cases. Let us start with $p'(X)$. Suppose the node which stores $p'(w + \nu)$ goes offline. A new comer can get $p'(w + \nu)$ by asking for $p'(w)$ and $p'(\nu)$, since

$$p'(w + \nu) = p'(w) + p'(\nu).$$

If we are instead using $p(X)$, when the node which stores $p(w+\nu)$ goes offline, then for any choice of $u, v \neq 0,\ u, v \in \mathbb{F}_8$, a new comer can ask $p(uw)$ and $p(v\nu)$, and then compute

$$u^{-1}p(uw) + v^{-1}p(v\nu) = p(w) + p(\nu).$$

Example 5 The HSRC codes described above implicitly assume a specific, fixed input size, determined by the choices of n, k, q on which the coding is to be performed. This is also the case for many other coding schemes such as Reed-Solomon codes. In real life, data objects may however come in an arbitrary size. Two heuristics deal with the consequent constraints - namely, zero padding (for object which is too small), and slicing (for a large object).

For instance, consider as object $\mathbf{o}$ a file of 4MB. We can cut $\mathbf{o}$ into 4 pieces of 1MB each, and directly use a $(15, 4)$ code. Alternatively, we can cut $\mathbf{o}$ into 4 slices $\mathbf{s}_1, \ldots, \mathbf{s}_4$, each of size 1MB. We now encode each $\mathbf{s}_i$ into a codeword $\mathbf{x}_i = (x_{i1}, \ldots, x_{i,15})$, using again the $(15, 4)$ code. The jth node then stores the jth encoded fragment x_{ij} $j = 1, \ldots, 15$ for all the slices. To get a point of comparison of the parameter values (n, k) used in this example, we note that the erstwhile peer-to-peer realization of Wuala claimed the use of a (517,100) code, which has a rate k/n of $100/517 \simeq 0.19$, while the $(15, 4)$ code has rate $4/15 \simeq 0.26$. Such a range of rates (and corresponding storage overheads) is typical in volatile environments as is characteristic of peer-to-peer systems.

A final observation we want to make here is that, in the following section, when we analyse the static resilience of a code, it determines the availability for one slice of the object, rather than the object itself. However, placing the encoded fragments of all the slices in a common pool of storage nodes - which is also practical in terms of managing meta-information - leads to the same availability of the object, as for the individual slices. Hence we do not distinguish the two in the rest of this paper, and consider that the code could be applied on any object, independently of its size. It has to be noted here that if the encoded fragments of different slices were to be placed among different set of nodes, this would however not hold true. This however is an issue we will not delve into any further, and is also not usually practiced due to practical system design considerations.

4 Static Resilience Analysis

The rest of the paper is dedicated to the analysis of the proposed homomorphic self-repairing codes. *Static resilience* of a distributed storage system is defined as the probability that an object, once stored in the system, will continue to stay available without any further maintenance, even when a certain fraction of individual member nodes of the distributed system become unavailable. We start the evaluation of the proposed scheme with a static resilience analysis, where we study how a stored object can be recovered using HSRCs, compared with traditional erasure codes, prior to considering the maintenance process, which will be done in Section 5.

Let p_{node} be the probability that any specific node is available. Then, under the assumptions that node availability is *i.i.d*, and no two fragments of the same object are placed on any same node, we can consider that the availability of any fragment is also *i.i.d* with probability p_{node}.

4.1 A code representation

Recall that using the above coding strategy, an object $\mathbf{o}$ of length M is decomposed into k fragments of length M/k:

$$\mathbf{o} = (\mathbf{o}_1, \ldots, \mathbf{o}_k),\ \mathbf{o}_i \in \mathbb{F}_{q^{M/k}},$$

which are further encoded into n fragments of same length:

$$\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n),\ \mathbf{x}_i \in \mathbb{F}_{q^{M/k}},$$

each of the encoded fragment $\mathbf{x}_i = p(\alpha_i)$ is given to a node to be stored. We thus have n nodes each possessing a q-ary vector of length M/k, corresponding to a system of n linear equations

$$\begin{pmatrix} \alpha_1 & \alpha_1^q & \alpha_1^{q^2} & \ldots & \alpha_1^{q^{k-1}} \\ \alpha_2 & \alpha_2^q & \alpha_2^{q^2} & \ldots & \alpha_2^{q^{k-1}} \\ \vdots & & \vdots & & \\ \alpha_n & \alpha_n^q & \alpha_n^{q^2} & \ldots & \alpha_n^{q^{k-1}} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{k-1} \end{pmatrix} = \begin{pmatrix} p(\alpha_1) \\ p(\alpha_2) \\ \vdots \\ p(\alpha_n) \end{pmatrix}.$$

If three rows are $\mathbb{F}_q$-linearly dependent, say rows $1, 2$ and 3, then

$$u(\alpha_1, \alpha_1^q, \alpha_1^{q^2}, \ldots, \alpha_1^{q^{k-1}}) + v(\alpha_2, \alpha_2^q, \alpha_2^{q^2}, \ldots, \alpha_2^{q^{k-1}})$$
$$= (\alpha_3, \alpha_3^q, \alpha_3^{q^2}, \ldots, \alpha_3^{q^{k-1}}),\ u, v \in \mathbb{F}_q,$$

which can be rewritten as

$$(u\alpha_1 + v\alpha_2, u\alpha_1^q + v\alpha_2^q, u\alpha_1^{q^2} + v\alpha_2^{q^2}, \ldots, u\alpha_1^{q^{k-1}} + v\alpha_2^{q^{k-1}})$$
$$= (u\alpha_1 + v\alpha_2, (u\alpha_1 + v\alpha_2)^q, (u\alpha_1 + v\alpha_2)^{q^2}, \ldots, (u\alpha_1 + v\alpha_2)^{q^{k-1}})$$

$$= (\alpha_3, \alpha_3^q, \alpha_3^{q^2}, \ldots, \alpha_3^{q^{k-1}}),\ u, v \in \mathbb{F}_q \iff u\alpha_1 + v\alpha_2 = \alpha_3.$$

Thus to understand the linear dependencies among the fragments owed by each of the n nodes, one can associate to the ith node the value α_i. Once all the α_i are written in a $\mathbb{F}_q$-basis, they can be represented as an $n \times M/k$ q-ary matrix

$$\mathbb{M} = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} \alpha_{1,1} & \ldots & \alpha_{1,M/k} \\ \vdots & & \vdots \\ \alpha_{n,1} & \ldots & \alpha_{n,M/k} \end{pmatrix} \tag{7}$$

with $\alpha_{i,j} \in \mathbb{F}_q$.

Example 6 In Example 3, by fixing as $\mathbb{F}_2$-basis $\{1, w, w^2, w^3\}$, we have for $n = 4$ that $\mathbb{M} = I_4$, the 4-dimensional identity matrix, since $\alpha_i = w^i,\ i = 0, \ldots, 3$, while for $n = 7$, it is

$$\mathbb{M}^T = \begin{pmatrix} 1\,0\,0\,1\,0\,1\,1 \\ 0\,1\,0\,1\,1\,0\,1 \\ 0\,0\,1\,0\,1\,1\,1 \\ 0\,0\,0\,0\,0\,0\,0 \end{pmatrix},$$

corresponding to $1, w, w^2, w^4, w^5, w^8, w^{10}$. Now in Example 4, by fixing as $\mathbb{F}_8$-basis $\{1, \nu, \nu^2, \nu^3\}$, we have

$$\mathbb{M}^T = \begin{pmatrix} u \\ v \\ 0 \\ 0 \end{pmatrix},\ u, v \in \mathbb{F}_8,\ (u, v) \neq 0.$$

Thus unavailability of a random node is equivalent to losing one linear equation, or a random row of the matrix $\mathbb{M}$. If multiple random nodes (say $n - x$) become unavailable, then the remaining x nodes provide x encoded fragments, which can be represented by an $x \times M/k$ sub-matrix $\mathbb{M}_x$ of $\mathbb{M}$. For any given combination of such x available encoded fragments, the original object can still be reconstructed if we can obtain at least k linearly independent rows of $\mathbb{M}_x$. This is equivalent to say that the object can be reconstructed if the rank of the matrix $\mathbb{M}_x$ is larger than or equal to k.

In the case the polynomial $p(X) = \sum_{i=0}^{k-1} p_i X^{2^i},\ p_i \in \mathbb{F}_q$ is chosen for encoding, the corresponding system of n linear equations is slightly different

$$\begin{pmatrix} \alpha_1 & \alpha_1^2 & \alpha_1^{2^2} & \ldots & \alpha_1^{2^{k-1}} \\ \alpha_2 & \alpha_2^2 & \alpha_2^{2^2} & \ldots & \alpha_2^{2^{k-1}} \\ \vdots & & \vdots & & \\ \alpha_n & \alpha_n^2 & \alpha_n^{2^2} & \ldots & \alpha_n^{2^{k-1}} \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{k-1} \end{pmatrix} = \begin{pmatrix} p(\alpha_1) \\ p(\alpha_2) \\ \vdots \\ p(\alpha_n) \end{pmatrix},$$

so that now, though it is still true that if three rows are $\mathbb{F}_q$-linearly dependent, say rows $1, 2$ and 3, then

$$u(\alpha_1, \alpha_1^2, \alpha_1^{2^2}, \ldots, \alpha_1^{2^{k-1}}) + v(\alpha_2, \alpha_2^2, \alpha_2^{2^2}, \ldots, \alpha_2^{2^{k-1}})$$

$$= (\alpha_3, \alpha_3^2, \alpha_3^{2^2}, \ldots, \alpha_3^{2^{k-1}}),\ u, v \in \mathbb{F}_q,$$

it is not true anymore that

$$(u\alpha_1 + v\alpha_2, u\alpha_1^2 + v\alpha_2^2, u\alpha_1^{2^2} + v\alpha_2^{2^2}, \ldots, u\alpha_1^{2^{k-1}} + v\alpha_2^{2^{k-1}})$$

$$= (u\alpha_1 + v\alpha_2, (u\alpha_1 + v\alpha_2)^2, (u\alpha_1 + v\alpha_2)^{2^2}, \ldots, (u\alpha_1 + v\alpha_2)^{2^{k-1}})$$

since $u^2 = u$, resp. $v^2 = v$ holds if and only if $u, v \in \mathbb{F}_2$. In this case, we have to analyze the matrix

$$\begin{pmatrix} \alpha_1 & \alpha_1^2 & \alpha_1^{2^2} & \ldots & \alpha_1^{2^{k-1}} \\ \alpha_2 & \alpha_2^2 & \alpha_2^{2^2} & \ldots & \alpha_2^{2^{k-1}} \\ \vdots & & & \vdots & \\ \alpha_n & \alpha_n^2 & \alpha_n^{2^2} & \ldots & \alpha_n^{2^{k-1}} \end{pmatrix} \tag{8}$$

directly.

Example 7 *Consider again Example 4, and suppose the two polynomials $p(X)$ and $p'(X)$ are both evaluated in $\alpha_1 = 1$, $\alpha_2 = \nu$, and $\alpha_3 = w+\nu$. Clearly $w\alpha_1 + \alpha_2 = \alpha_3$. Consequently, when evaluating the polynomial $p(X)$ in $\alpha_1, \alpha_2, \alpha_3$, we get as part of the system of linear equations the following 3 rows:*

$$(1, 1, 1, 1), (\nu, \nu^8, \nu^{64}, \nu^{512})$$

and

$$(w + \nu, (w + \nu)^8, (w + \nu)^{64}, (w + \nu)^{512}) =$$

$$(w + \nu, w^8 + \nu^8, w^{64} + \nu^{64}, w^{512} + \nu^{512}).$$

Clearly the three rows are $\mathbb{F}_8$-linearly dependent. If now instead the polynomial $p'(X)$ is similarly evaluated, we obtain

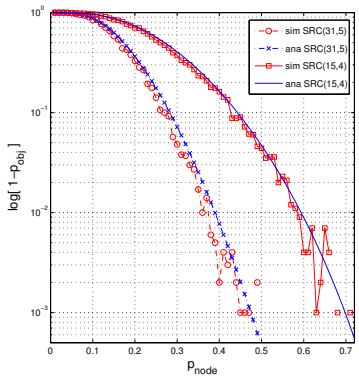
$$(1, 1, 1, 1), (\nu, \nu^2, \nu^4, \nu^8)$$

and

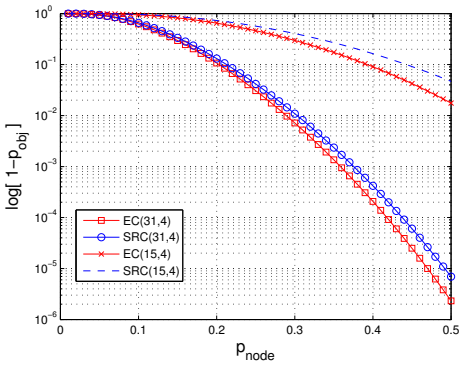
$$(w + \nu, (w + \nu)^2, (w + \nu)^4, (w + \nu)^8).$$

This time the dependencies disappear, since

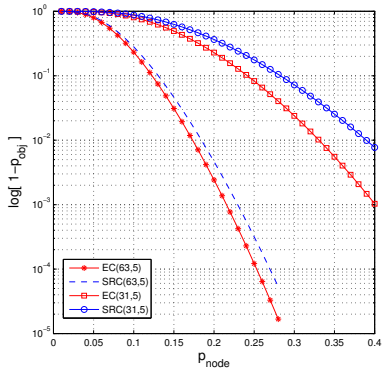
$$w + \nu^2 \neq (w + \nu)^2 = w^2 + \nu^2.$$



(a) Validation of the static resilience analysis



(b) Comparison of SRC with EC



(c) Comparison of SRC with EC

Fig. 1 Static resilience of homomorphic self-repairing codes (HSRC) for $q = 2$: Validation of analysis, and comparison with MDS erasure codes (EC), where the y-axis is in logscale, depicting the object unavailability $(1 - p_{obj})$.

4.2 Probability of object retrieval

Consider a $(q^d - 1) \times d$ q-ary matrix for some $d > 1$, with distinct rows, no all zero row, and thus rank d. The case of interest for us is $d = M/k$, since $\mathbb{M}$ is an $(q^{M/k} - 1) \times (M/k)$ matrix. If we remove some of the rows uniformly randomly with some probability $1 - p_{node}$, then we are left with a $x \times d$ sub-matrix - where x is binomially distributed. We define $R(x, d, r)$ as the number of $x \times d$

sub-matrices with rank r, voluntarily including all the possible permutations of the rows in the counting.

Lemma 2 *Let $R(x,d,r)$ be the number of $x \times d$ sub-matrices with rank r of a tall $(q^d-1)\times d$ matrix of rank d. We have that $R(x,d,r)=0$ when (i) $r=0$, (ii) $r>x$, (iii) $r=x$, with $x>d$, or (iv) $r<x$ but $r>d$. Then, counting row permutations:*

$$R(x,d,r)=\prod_{i=0}^{r-1}(q^d-q^i) \text{ if } r=x, x\leq d,$$

and for $r<x$ with $r\leq d$:

$$R(x,d,r)=R(x-1,d,r-1)(q^d-q^{r-1})+R(x-1,d,r)(q^r-x).$$

Proof There are no non-trivial matrix with rank $r=0$. When $r>x$, $r=x$ with $x>d$, or $r<x$ but $r>d$, $R(x,d,r)=0$ since the rank of a matrix cannot be larger than the smallest of its dimensions.

For the case when $r=x$, with $x\leq d$, we deduce $R(x,d,r)$ as follows. To build a matrix $\mathbb{M}_x$ of rank $x=r$, the first row can be chosen from any of the q^d-1 rows in $\mathbb{M}$, and the second row should not be a multiple of the first row, which gives q^d-2 choices. The third row needs to be $\mathbb{F}_q$-linearly independent from the first two rows. Since there are q^2 linear combinations of the first two rows, which includes the all zero vector which is discarded, we obtain q^d-q^2 choices. In general, the $(i+1)$st row can be chosen from q^d-q^i options that are linearly independent from the i rows that have already been chosen. We thus obtain $R(x,d,r)=\prod_{i=0}^{r-1}(q^d-q^i)$ for $r=x$, $x\leq d$.

For the case where $r<x$ with $r\leq d$, we observe that $x\times d$ matrices of rank r can be inductively obtained by either (I) adding a linearly independent row to a $(x-1)\times d$ matrix of rank $r-1$, or (II) adding a linearly dependent row to a $(x-1)\times d$ matrix of rank r. We use this observation to derive the recursive relation

$$R(x,d,r)=R(x-1,d,r-1)(q^d-q^{r-1})+R(x-1,d,r)(q^r-x),$$

where $q^d-1-(q^{r-1}-1)$ counts the number of linearly independent rows that can be added, and $q^r-1-(x-1)$ is on the contrary the number of linearly dependent rows.

We now remove the permutations that we counted in the above analysis by introducing a suitable normalization.

Corollary 2 *Let $\rho(x,d,r)$ be the fraction of sub-matrices of dimension $x\times d$ with rank r out of all possible sub-matrices of the same dimension. Then*

$$\rho(x,d,r)=\frac{R(x,d,r)}{\sum_{j=0}^{d}R(x,d,j)}=\frac{R(x,d,r)}{C_x^{q^d-1}x!}.$$

In particular

$$\rho_x(d) = \sum_{r=k}^{d} \rho(x, d, r) \tag{9}$$

is the conditional probability that the stored object can be retrieved by contacting an arbitrary x out of the n storage nodes.

Proof It is enough to notice that there are $C_x^{q^d-1}$ ways to choose x rows out of the possible $q^d - 1$ options. The chosen x rows can be ordered in $x!$ permutations.

In particular, when the rank is at least k, the object can be retrieved.

We now put together the above results to compute the probability p_{obj} of an object being recoverable when using an HSRC(n, k) code to store a length M object made of k fragments encoded into n fragments each of length M/k.

Corollary 3 *Using an* HSRC(n, k)*, the probability p_{obj} of recovering the object is*

$$p_{obj} = \sum_{x=k}^{n} \sum_{r=k}^{d} \rho(x, d, r) C_x^n p_{node}^x (1 - p_{node})^{n-x},$$

where $d = \log_q n + 1$.

Proof If $n = n_{max} = q^{M/k} - 1$, we apply Lemma 2 and Corollary 2 with $d = M/k$. If $n = q^i - 1$, for some integer $i \leq M/k$ such that $n \geq q^k - 1$, then $\mathbb{M}$ has $M/k - i$ columns which are either all zeros or all ones vectors, as shown on Example 6. Thus the number of its sub-matrices of rank r is given by applying Lemma 2 on the matrix obtained by removing these redundant columns.

We validate the analysis with simulations, and as can be observed from Figure 4.1, we obtain a precise match.

To conclude this analysis, let us get back to Example 4 and notice that the static resilience analysis derived above holds for the encoding via the polynomial $p(X)$. It would not be the case if $p'(X)$ were used, in which case only the $\mathbb{F}_2$-linear dependencies should be kept.

4.3 Comparison with standard erasure codes

While there is marginal deterioration of static resilience using SRC with respect to MDS codes (as compared in Fig. 1, and to be discussed soon after), we first elaborate how SRC differs fundamentally from MDS codes by looking at the conditional probability that the stored object can be retrieved by contacting an arbitrary x out of the n storage nodes.

For (n, k) MDS erasure codes, ρ_x is a deterministic and binary value equal to one for $x \geq k$, and zero for smaller x. For self-repairing codes, the value is

probabilistic. In Fig. 2 we show for our toy example $HSRC(31, 5)$ the probability that the object can be retrieved by contacting arbitrary x nodes, i.e., ρ_x, where the values of ρ_x for $x \geq k$ were computed from (9).[3] This can alternatively be interpreted as the probability that the object is retrievable despite precisely $n-x$ random failures, and only x random storage nodes are available.

In particular, if any five storage nodes are randomly picked, it is likely that the object cannot be reconstructed with a probability 0.5096, while if any seven random nodes are picked, this probability decreases to 0.0757, while, if thirteen or more random nodes are picked, then the object can certainly be reconstructed. In contrast, for MDS codes, the object will be retrievable from the data available at any arbitrary five nodes.

Of-course, this rather marginal sacrifice (we will next compare HSRC's static resilience with MDS erasure codes to demonstrate the marginality) provides HSRC an incredible amount of self-repairing capability. Given that a practical system will carry out repairs rather frequently, and HSRC in fact allows very cheap repairs, a system using HSRC will be more easily and cheaply maintained, and hence be reliable - particularly be avoiding multiple failures to cumulate.

Likewise, for data access and reconstruction, in practice, storage nodes will be accessed in a planned manner, rather than randomly. There are in fact $C_k^n \rho_k$ (i.e., 83324 for HSRC$(31, 5)$) unique subsets of precisely k storage nodes that allows reconstruction. Hence, in practice, object access overheads will not be different than when using a MDS coding based scheme.

Let us now compare HSRC against standard MDS erasure codes in terms of the effective static resilience. If we use a (n, k) MDS erasure code, then the probability that the object is recoverable when each individual storage node may fail i.i.d. with probability $1 - p_{node}$ is:

$$p_{obj} = \sum_{i=k}^{n} C_i^n p_{node}^i (1 - p_{node})^{n-i}.$$

Note that MDS codes may not exist for specific arbitrary choice of n and k. However, for the sake of fair comparison, this formula and the following plots are provided as if they were to exist.

In Figures 4.1 and 4.1, we compare the static resilience achieved using the proposed homomorphic SRC with that of MDS erasure codes.

In order to achieve the self-repairing property in SRC, it is obvious that it is necessary to introduce extra 'redundancy' in its code structure, but we notice from the comparisons that this overhead is in fact marginal. By marginal, we mean that since most real-world systems typically operate with few (but not many) faults at any given time, the conditional probability of retrieving a data stored using SRC or a MDS code are the same for a range of typical number of faults. Furthermore, for the same storage overhead n/k, the overall static resilience of SRC is only slightly lower than that of EC, and for a fixed k, as the value of n increases, SRC's static resilience gets very close to that of

[3] ρ_x is zero for $x < k$ for HSRC also.

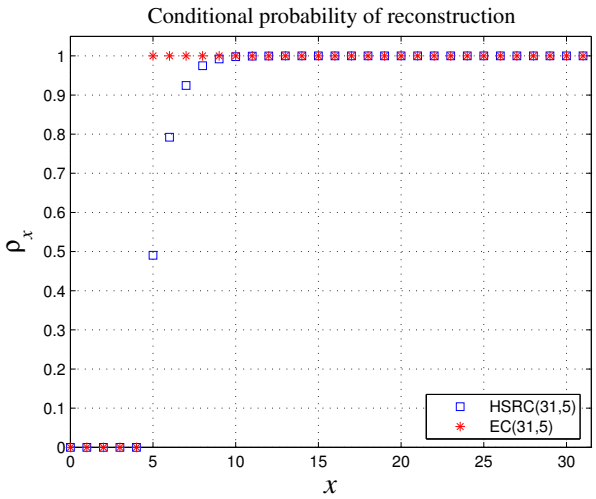


Fig. 2 Comparison of the probability of reconstruction of object using encoded data in x random storage nodes

EC. Finally, even for low storage overheads, with relatively high p_{node}, the probability of object availability is indeed 1. In any storage system, there will be a maintenance operation to replenish lost fragments (and hence, the system will operate for high values of p_{node}).

We will further see in the next section that SRCs have low maintenance overheads. These make SRCs a practical coding scheme for networked storage.

5 Communication overheads of self-repair

In the previous section we studied the probability of recovering an object if it so happens that only p_{node} fraction of nodes which had originally stored the encoded fragments continue to remain available, while lost redundancy is yet to be replenished. Such a situation may arise either because a lazy maintenance mechanism (such as, in [2]) is applied, which triggers repairs only when redundancy is reduced to certain threshold, or else because of multiple correlated failures before repair operations may be carried out. We will next investigate the communication overheads in such scenarios, emphasizing on those HSRC with an XOR-like structure (that is, retaining $\mathbb{F}_2$-linearity). Note that this is really the regime in which we need an analysis, since in absence of correlated failures, and assuming that an eager repair strategy is applied, whenever one encoded block is detected to be unavailable, it is immediately replenished. The proposed HSRC ensures that this one missing fragment can be replenished by obtaining only two other (appropriate) encoded fragments, thanks to the HSRC subspace structure.

Definition 4 *The diversity δ of SRC is defined as the number of mutually exclusive pairs of fragments which can be used to recreate any specific fragment.*

In other words, the notion of diversity captures the number of local repairs that can be performed in the case of several (correlated) failures. This is important for any system that needs to support local repairs of at least two failures.

In Example 3, it can be seen easily that $\delta = 3$. Let us assume that $p(w)$ is missing. Any of the three exclusive fragment pairs, namely $((p(1), p(w^4))$; $(p(w^2), p(w^5))$ or $(p(w^8), p(w^{10}))$ may be used to reconstruct $p(w)$. See Table 2 for other examples. In Example 4 where the encoding is done using $p'(X)$, the diversity is $\delta = 31$. Indeed, every encoded fragment is of the form $p'(u + \nu v)$, $u, v \in \mathbb{F}_8$, so that for every $u' + \nu v'$, $u', v' \in \mathbb{F}_8$, we have that the pair $(p'(u' + \nu v'), p'((u' + u) + \nu(v' + v)))$ can be used to reconstruct $p'(u + \nu v)$, since $p'(u' + \nu v') + p'((u' + u) + \nu(v' + v)) = p'(u + \nu v)$, and the fragment $p'((u' + u) + \nu(v' + v))$ is indeed present in the network since $u', v' \in \mathbb{F}_8$ and u, v run through every element in $\mathbb{F}_8$ but for the pair $(0, 0)$.

Lemma 3 *The diversity δ of a* HSRC(n, k) *is* $(n - 1)/2$.

Proof We have that $n = q^d - 1$ for some suitable d. The polynomial $p(x)$ is evaluated in $\alpha = \sum_{i=0}^{d-1} a_i w^i$, where $a_i \in \mathbb{F}_q$ and $(a_0, ..., a_{d-1})$ takes all the possible q^d values, but for the whole zero one. Thus for every α, we can create the pairs $(\alpha + \beta, \beta)$ where β takes $q^d - 2$ possible values, that is all values besides 0 and α. This gives $q^d - 2$ (which is equal to $n - 1$) pairs, but since pairs $(\alpha + \beta, \beta)$ and $(\beta, \alpha + \beta)$ are equivalent, we have $(n - 1)/2$ distinct such pairs.

An interesting property of SRC can be inferred from its diversity.

Corollary 4 *For a Homomorphic SRC, if at least $(n + 1)/2$ fragments are available, then for any of the unavailable fragments, there exists some pair of available fragments which is adequate to reconstruct the unavailable fragment.*

Proof Consider any arbitrary missing fragment α. If up to $(n-1)/2$ fragments were available, in the worst case, these could belong to the $(n-1)/2$ exclusive pairs. However, if an additional fragment is available, it will be paired with one of these other fragments, and hence, there will be at least one available pair with which α can be reconstructed.

5.1 Overheads of recreating one specific missing fragment

Recall that x is defined as the number of fragments of an object that are available at a given time point. For any specific missing fragment, any one of the corresponding mutually exclusive pairs is adequate to recreate the said fragment. From Corollary 4 we know that if $x \geq (n+1)/2$ then two downloads

are enough. Otherwise, we need a probabilistic analysis. Both nodes of a specific pair are available with probability $(x/n)^2$. The probability that only two fragments are enough to recreate the missing fragment is $p_2 = 1-(1-(x/n)^2)^\delta$.

If two fragments are not enough to recreate a specific fragment, it may still be possible to reconstruct it with larger number of fragments. A loose upper bound can be estimated by considering that if 2 fragments are not adequate, k fragments need to be downloaded to reconstruct a fragment,[4] which happens with a probability $1 - p_2 = (1 - (x/n)^2)^\delta$.

Thus the expected number D_x of fragments that need to be downloaded to recreate one fragment, when x out of the n encoded fragments are available, can be determined as:

$$D_x = 2 \text{ if } x \geq (n+1)/2$$
$$D_x < 2p_2 + k(1-p_2) \text{ if } x < (n+1)/2.$$

5.2 Overhead of recreating all missing fragments

Above, we studied the overheads to recreate one fragment. All the missing fragments may be repaired, either in parallel (distributed in different parts of the network) or in sequence. If all missing fragments are repaired in parallel, then the total overhead D_{prl} of downloading necessary fragments is:

$$D_{prl} = (n-x)D_x.$$

If they are recreated sequentially, then the overhead D_{seq} of downloading necessary fragments is:

$$D_{seq} = \sum_{i=x}^{n} D_i.$$

In order to directly compare the overheads of repair for different repair strategies - eager, or lazy parallelized and lazy sequential repairs using SRC, as well as lazy repair with traditional erasure codes, consider that lazy repairs are triggered when a threshold $x = x_{th}$ of available encoded fragments out of n is reached. If eager repair were used for SRC encoded objects, download overhead of

$$D_{egr} = 2(n - x_{th})$$

is incurred. Note that, when SRC is applied, the aggregate bandwidth usage for eager repair as well as both lazy repair strategies is the same, assuming that the threshold for lazy repair $x_{th} \geq (n+1)/2$.

In the setting of traditional erasure codes, let us assume that one node downloads enough (k) fragments to recreate the original object, and recreates

[4] Note than in fact, often fewer than k fragments will be adequate to reconstruct a specific fragment.

one fragment to be stored locally, and also recreates the remaining $n - x_{th} - 1$ fragments, and stores these at other nodes. This leads to a total network traffic:

$$D_{EClazy} = k + n - x_{th} - 1.$$

Eager strategy using traditional erasure codes will incur k downloads for each repair, which is obviously worse than all the other scenarios, so we ignore it in our comparison.

Note that if less than half of the fragments are unavailable, as observed in Corollary 4, downloading two blocks is adequate to recreate any specific missing fragment. When too many blocks are already missing, applying a repair strategy analogous to traditional erasure codes, that of downloading k blocks to recreate the whole object, and then recreate all the missing blocks is logical. That is to say, the benefit of reduced maintenance bandwidth usage for SRC (as also of other recent techniques like RGC) only makes sense under a regime when not too many blocks are unavailable. Let us define x_c as the critical value, such that if the threshold for lazy repair in traditional erasure codes x_{th} is less than this critical value, then, the aggregate fragment transfer traffic to recreate missing blocks will be less using the traditional technique (of downloading k fragments to recreate whole object, and then replenish missing fragments) than by using SRC. Recall that for $x \geq (n+1)/2$, $D_{egr} = D_{prl} = D_{seq}$. One can determine x_c as follows. We need $D_{egr} \leq D_{EClazy}$, implying that

$$2n - 2x_c \leq n - 1 + k - x_c \Rightarrow x_c = n + 1 - k.$$

Figure 3 shows the average amount of network traffic to transfer data from live nodes per lost encoded fragment when the various lazy variants of repair are used, namely parallel (γ_{prl}) and sequential (γ_{seq}) repairs with SRC, and (by default, sequential) repair (γ_{eclazy}) when using EC. MSR RGCs are also shown on this figure - see γ_{MSRGC}, where d is the number of live nodes contacted during repair. We also note that MBR RGCs would incur less bandwidth per repair than MSR RGCs. However, an individual MBR encoded piece itself has some inherent redundancy, and does not carry the same amount of information as a standard (Reed-Solomon) MDS erasure encoded piece or HSRC encoded piece. Thus, we have restricted our comparison with only MSR RGCs.

The *x-axis* represents the threshold x_{th} for lazy repair, such that repairs are triggered only if the number of available blocks for an object is not more than x_{th}. Use of an eager approach with SRC incurs a constant overhead of two fragments per lost block. Note that there are other messaging overheads to disseminate necessary meta-information (e.g., which node stores which fragment), but we ignore these in the figure, considering that the objects being stored are large, and data transfer of object fragments dominates the network traffic. This assumption is reasonable, since for small-objects, it is well known that the meta-information storage overheads outweigh the benefits of using erasure codes, and hence erasure coding is impractical for small objects.

There are several implications of the above observed behaviors. To start with, we note that an engineering solution like lazy repair which advocates

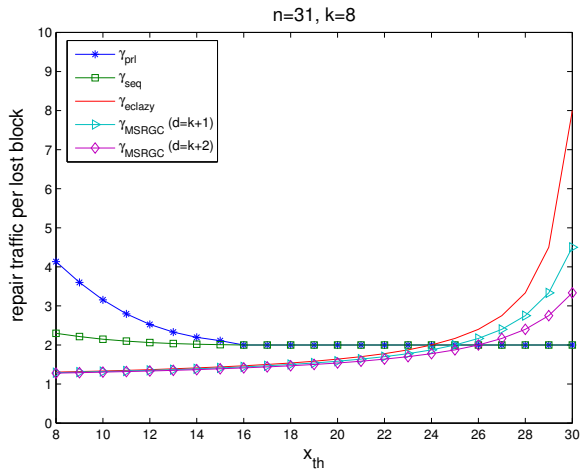


Fig. 3 Average traffic normalized with M/k per lost block for various choices of x_{th}.

waiting before repairs are triggered, amortizes the repair cost per lost fragment, and is effective in reducing total bandwidth consumption and outperforms SRC (in terms of total bandwidth consumption), provided the threshold of repair x_{th} is chosen to be lower than x_c. This is in itself not surprising. However, for many typical choices of (n, k) in deployed systems such as $(16, 10)$ in Cleversafe [3], or $(517, 100)$ in Wuala [13], a scheme like SRC is practical. In the former scenario, x_c is too low, and waiting so long makes the system too vulnerable to any further failures (i.e., poor system health). In the later scenario, that is, waiting for hundred failures before triggering repairs seems both unnecessary, and also, trying to repair 100 lost fragments simultaneously will lead to huge bandwidth spikes.[5]

Using SRC allows for a flexible choice of either an eager or lazy (but with much higher threshold x_{th}) approaches to carry out repairs, where the repair cost per lost block stays constant for a wide range of values (up till $x_{th} \geq (n+1)/2$). Such a flexible choice makes it easier to also benefit from the primary advantage of lazy repair in peer-to-peer systems, namely, to avoid unnecessary repairs due to temporary churn, without the drawbacks of (i) having to choose a threshold which leads to system vulnerability or (ii) choose a much higher value of n in order to deal with such vulnerability, and (iii) have spiky bandwidth usage.

5.3 Maximal distance separability & minimum storage point

To conclude the discussion on the cost of repair, this subsection gives some points of comparison between the now well known RGC and the newly introduced SRC. The theory underlying regenerating codes exposes an interesting

[5] A storage system's vulnerability to further failures, as well as spiky bandwidth usage are known problems of lazy repair strategies [11].

trade-off between the storage and repair bandwidth overheads for maximal distance separable codes - where the data is encoded and stored over n nodes, and encoded data stored at any arbitrary k of these storage nodes allows reconstruction of the whole object.

Suppose that each node has a storage capacity of α, i.e., the size of the encoded data block stored at a node is of the size α. When one data block needs to be regenerated, a new node contacts d live nodes, and downloads β amount of data from each of the contacted nodes (referred to as the bandwidth capacity of the connections between any node pair). By considering an information flow from the source to the data collector, a trade-off between the nodes' storage capacity and bandwidth is computed through a min-cut bound. This analysis determines two interesting constraints. Firstly, regeneration of a lost node is feasible only when at least k live nodes are contacted, i.e. $d \geq k$. Secondly, it determines a trade-off curve between the storage overhead per node α and the bandwidth per regeneration $d\beta$. One extreme of this trade-off curve corresponds to the smallest feasible value of α, which is M/k (the other one being the smallest feasible value of β, called the minimal bandwidth repair point (MBR)). We note that this storage overhead corresponds to any optimal encoding scheme which aims to reconstruct the object using no more than k encoded blocks. This point on the trade-off curve is called the *minimum storage repair point* (MSR), determining the minimal bandwidth requirement for regeneration according the min-cut max-flow arguments of information flow as follows:

$$(\alpha_{MSR}, \beta_{MSR}) = \left(\frac{M}{k}, \frac{M}{k(d-k+1)} \right).$$

A similar minimum storage point, computed using the same type of arguments, is available for collaborative RGCs, and takes the form

$$\alpha = \frac{M}{k}, \ \beta = \beta' = \frac{M}{k} \frac{1}{d-k+t}$$

where β'_{MSR} denotes the bandwidth used for cooperation and t is the number of new nodes regenerating together and in cooperation with each other (thus, t could be interpreted as the number of failures triggering lazy, collaborative repair).

Since encoded blocks in HSRC(n, k) codes are also of size M/k, a meaningful comparison is possible corresponding to the minimal storage (MSR) point. We note that firstly, HSRC achieves a $d << k$, which breaches the $d \geq k$ constraint of RGCs. Furthermore, we notice from Figure 3 that HSRC can carry out regenerations with less bandwidth per repair than RGCs, for certain number of faults, and certain RGC parameter choices.

At first sight, this may seem counter-intuitive, given that the max-flow min-cut analysis establishes hard achievability constraints. But recall that these constraints were determined under the assumption of maximal distance separability of the resulting code. Thus to say, the repair advantage of HSRC is obtained by relaxing the MDS constraint, which, as discussed in Section 4.3, has marginal practical drawbacks or overheads.

Finally, it is worth pointing out the significance of the choice of d. A typical value of k as used in volatile environments like peer-to-peer systems is about 100 (a choice advocated in erstwhile Wuala system), this means that the number of nodes contacted for one repair is more than a hundred, whereas HSRC in contrast can repair one node by communicating with only two nodes.

6 Other practical implications: A qualitative discussion

So far we have demonstrated that by embracing the self-repairing properties, significant reduction in the aggregate bandwidth used for repairs is achieved. This overhead reduction is with respect to not only traditional erasure codes, but under certain regimes, also in comparison to other 'optimal' storage centric codes such as regenerating codes. While repair bandwidth overhead reduction was the explicit motivation for designing self-repairing codes, the code properties have another natural and desirable consequence - in presence of multiple failures, SRC allows for fast and parallel repairs. We will elaborate this property with an example in Subsection 6.1 below.

Apart the lack of maximum distance separability (MDS) property, another possible critique of HSRC is that it is not a systematic code, that is, pieces of the object are not present uncoded. We have already argued that the original design goal of the self-repairing properties themselves are mutually exclusive with the MDS property, but based on quantitative arguments (see Section 4.3), we concluded that this has marginal impact on the resilience or storage overheads of the proposed code. We note that the systematic code property is not necessarily and completely exclusive of the cardinal self-repairing code properties. Indeed, a different construct of self-repairing code [19] based on very different mathematical properties, that of projective geometry, has been shown to have systematic-like features. We further comment the issue of systematic pieces in Subsection 6.2.

6.1 Fast & parallel repairs with HSRC

We observed in the previous section that while SRC is effective in significantly reducing bandwidth usage to carry out maintenance of lost redundancy in coding based distributed storage systems, depending on system parameter choices, an engineering solution like lazy repair while using traditional EC may (or not) outperform SRC in terms of total bandwidth usage, even though using lazy repair with EC entails several other practical disadvantages.

A final advantage of SRC which we further showcase next is the possibility to carry out repairs of different fragments independently and in parallel (and hence, quickly). If repair is not fast, it is possible that further faults occur during the repair operations, leading to both performance deterioration as well as, potentially, loss of stored objects.

Consider the following scenario for ease of exposition: Assume that each node in the storage network has an uplink/downlink capacity of 1 (coded)

fragment	suitable pairs to reconstruct
$p(1)$	$(p(w^7), p(w^9));(p(w^{11}), p(w^{12}))$
$p(w)$	$(p(w^7), p(w^{14}));(p(w^8), p(w^{10}))$
$p(w^2)$	$(p(w^7), p(w^{12}));(p(w^9), p(w^{11}));(p(w^{12}), p(w^{10}))$
$p(w^3)$	$(p(w^8), p(w^{13}));(p(w^{10}), p(w^{12}))$
$p(w^4)$	$(p(w^9), p(w^{14}));(p(w^{11}), p(w^{13}))$
$p(w^5)$	$(p(w^7), p(w^{13}));(p(w^{12}), p(w^{14}))$
$p(w^6)$	$(p(w^7), p(w^{10}));(p(w^8), p(w^{14}))$

Table 3 Scenario: Seven fragments $p(1), \ldots, p(w^6)$ are missing

node	$p(w^0)$	$p(w^1)$	$p(w^2)$	$p(w^3)$	$p(w^4)$	$p(w^5)$	$p(w^6)$
Time 1	$p(w^7)$	$p(w^8)$	$p(w^9)$	$p(w^{13})$	$p(w^{11})$	$p(w^{12})$	$p(w^{10})$
Time 2	$p(w^9)$	$p(w^{10})$	$p(w^{11})$	$p(w^8)$	$p(w^{13})$	$p(w^{14})$	$p(w^7)$

fragment per unit time. Further assume that the network has relatively (much) larger aggregate bandwidth. Such assumptions correspond reasonably with various networked storage system environments.

Consider that for the Example 3, originally n was chosen to be n_{max}, that is to say, a HSRC$(15, 3)$ was used. Because of some reasons (e.g., lazy repair or correlated failures), let us say that seven encoded fragments, namely $p(1), \ldots, p(w^6)$ are unavailable while fragments $p(w^7)...p(w^{15})$ are available. Table 3 enumerates possible pairs to reconstruct each of the missing fragments.

A potential schedule to download the available blocks at different nodes to recreate the missing fragments is as follows: In first time slot, $p(w^{11})$, $p(w^{10})$, $p(w^{12})$, nothing, $p(w^{13})$, $p(w^7)$ and $p(w^8)$ are downloaded separately by seven nodes trying to recreate each of $p(1), \ldots, p(w^6)$ respectively. In second time slot $p(w^{12})$, $p(w^8)$, $p(w^7)$, $p(w^{10})$, $p(w^{11})$, $p(w^{13})$ and $p(w^{14})$ are downloaded. Note that, besides $p(w^3)$, all the other missing blocks can now already be recreated. In third time slot, $p(w^{12})$ can be downloaded to recreate it. Thus, in this example, six out of the seven missing blocks could be recreated within the time taken to download two fragments, while the last block could be recreated in the next time round, subject to the constraints that any node could download or upload only one block in unit time.

Even if a full copy of the object (hybrid strategy [25]) were to be maintained in the system, with which to replenish the seven missing blocks, it would have taken seven time units. While, if no full copy was maintained, using traditional erasure codes would have taken at least nine time units.

This example demonstrates that SRC allows for fast reconstruction of missing blocks. Orchestration of such distributed reconstruction to fully utilize this potential in itself poses interesting algorithmic and systems research challenges which we intend to pursue as part of future work.

6.2 On HSRC not being a systematic code

One can immediately read partial contents of the stored object from systematic encoded blocks. The fact that HSRC is not a systematic code (every encoded

fragment contains information about every piece of data) makes the object retrieval more costly: this is basically decoding. However, unlike in a classical communication scenario where decoding has to be done with whatever corrupted data is available, the situation is slightly different in archival storage: thanks to the repair property, it is possible to have a privileged set of encoded fragments to be used for decoding, and if some are missing during object retrieval, they can be repaired first. The set of encoded fragments to decode can be chosen for being closer to systematic blocks than random blocks, or precomputed computations can be made available to ease the decoding. Optimizing the decoding process is one of the future directions of the presented work. As a final note, we will like to mention that for deep archival (cold storage) applications, read operations are extremely infrequent, but persistence of the data is critical even in the presence of frequent failures, and significantly cheap repairs is a reasonable trade-off.

7 Conclusion

We considered the problem of dependability in networked distributed storage systems, focusing on fault-tolerance and maintainability. We propose a family of locally repairable erasure codes, called self-repairing codes, which are designed by taking into account specifically the characteristics of distributed networked storage systems. Self-repairing codes achieve excellent properties in terms of maintenance of lost redundancy in the storage system, most importantly: (i) low-bandwidth consumption for repairs (with flexible/somewhat independent choice of whether an eager or lazy repair strategy is employed), (ii) parallel and independent (thus very fast) replenishment of lost redundancy. When compared to MDS erasure codes, the self-repairing property is achieved by marginally compromising on static resilience for same storage overhead, or conversely, utilizing marginally more storage space to achieve equivalent static resilience. This paper provides the theoretical foundations for SRCs, and shows its potential benefits for distributed storage. There are several algorithmic and systems research challenges in harnessing SRCs in distributed storage systems, e.g., design of efficient decoding algorithms, or placement of encoded fragments to leverage on network topology to carry out parallel repairs, which are part of our ongoing and future work.

Acknowledgment

This work has been supported by the MoE Tier-2 grant MOE2013-T2-1-068 “eCODE: Erasure Codes for Datacenter Environments”.

References

1. L. Buttyán, L. Czap, I. Vajda, "Detection and Recovery from Pollution Attacks in Coding-Based Distributed Storage Schemes", *IEEE Transactions on Dependable and Secure Computing*, Vol. 8, No. 6, November - December 2011.
2. R. Bhagwan, K. Tati, Y. Cheng, S. Savage, G. Voelker, "Total recall: System support for automated availability management", *Networked Systems Design and Implementation (NSDI)*, 2004.
3. *http://www.cleversafe.org/dispersed-storage/configurations*
4. A. G. Dimakis, P. Brighten Godfrey, Y. Wu, M. O. Wainwright, K. Ramchandran, "Network Coding for Distributed Storage Systems", *IEEE Trans. on Information Theory*, vol. 56, no. 9, September 2010.
5. A. Datta, K. Aberer, "Internet-Scale Storage Systems under Churn – A Study of the Steady-State using Markov Models", *Peer-to-Peer Computing (P2P)*, 2006.
6. A. Duminuco, E. Biersack, "Hierarchical Codes: How to Make Erasure Codes Attractive for Peer-to-Peer Storage Systems", *Peer-to-Peer Computing (P2P)*, 2008.
7. A. Duminuco, E.W. Biersack, "A Practical Study of Regenerating Codes for Peer-to-Peer Backup Systems", *Intl. Conference on Distributed Computing Systems (ICDCS)*, 2009.
8. S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems", *Allerton conf. on comm. control and computing*, 2010.
9. S. Gaonkar, K. Keeton, A. Merchant, W. H. Sanders, " Designing Dependable Storage Solutions for Shared Application Environments", *IEEE Transactions on Dependable and Secure Computing*, Vol. 7, No 4, October - December 2010.
10. A.-M. Kermarrec, N. Le Scouarnec, G. Straub, "Beyong Regenerating Codes", Technical Report, 10 Sept 2010.
11. X. Liu, A. Datta, "Redundancy Maintenance and Garbage Collection Strategies in Peer-to-Peer Storage Systems", *Intl. Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)* 2009.
12. P. Gopalan, C. Huang, H. Simitchi and S. Yekhanin, 'On the locality of codeword symbols", *IEEE Trans. on Inform. Theory, vol. 58, no. 11, pp. 6925-6934, Nov. 2012.*
13. D. Grolimund, "Wuala - A Distributed File System", Google Tech Talk *http://www.youtube.com/watch?v=3xKZ4KGkQY8*
14. E. Haytaoglu, M. E. Dalkilic, "Homomorphic Minimum Bandwidth Repairing Codes", *Information Sciences and Systems 2013, Lecture Notes in Electrical Engineering*, vol. 264, 2013, pp. 339-348.
15. H. Hollmann, "Storage codes – coding rate and repair locality", *ICNC 2013.*
16. C. Huang, M. Chen, J. Li, "Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems", *NCA 2007*
17. G. M. Kamath, N. Prakash, V. Lalitha, P. Vijay Kumar, "Codes with Local Regeneration", in the proceedings of the Workshop on Information Theory and Applications (ITA), 2013. Available at `arXiv:1211.1932`, Nov. 2012.
18. F. Oggier, A. Datta, "Self-repairing Homomorphic Codes for Distributed Storage Systems", *INFOCOM 2011.*
19. F. Oggier, A. Datta, "Self-Repairing Codes for Distributed Storage - A Projective Geometric Construction", *ITW 2011.*
20. D.S. Papailiopoulos, A.G. Dimakis, "Locally Repairable Codes", *ISIT 2012*
21. K. V. Rashmi, N. B. Shah, P. V. Kumar and K. Ramchandran, "Explicit Construction of Optimal Exact Regenerating Codes for Distributed Storage", *Allerton Conf. on Control, Computing and Comm.* 2009.
22. K. V. Rashmi, N. B. Shah, P. V. Kumar, "Optimal Exact-Regenerating Codes for Distributed Storage at the MSR and MBR Points via a Product-Matrix Construction", *IEEE Transactions on Information Theory*, vol. 57, no. 8, Aug. 2011.
23. K. V. Rashmi, N. B. Shah and P. V. Kumar, "Enabling Node Repair in Any Erasure Code for Distributed Storage", *IEEE Int. Symp. on Inform. Theory*, 2011.
24. I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields", *Journal of the Society for Industrial and Appl. Mathematics*, no 2, vol. 8, SIAM, 1960.

25. R. Rodrigues and B. Liskov, "High Availability in DHTs: Erasure Coding vs. Replication", *Workshop on Peer-to-Peer Systems (IPTPS)* 2005.
26. K. W. Shum, "Cooperative Regenerating Codes for Distributed Storage Systems", *ICC 2011.*
27. K. W. Shum, Y. Hu, "Cooperative Regenerating Codes", *arXiv:1207.6762*
28. N. Silberstein, A.S. Rawat, O. Koyluoglu, S. Vishwanath, "Optimal Locally Repairable Codes via Rank-Metric Codes", `arXiv:1301.6331`, Oct. 2012.
29. I. Tamo, Z. Wang, and J. Bruck, "MDS array codes with optimal rebuilding", *IEEE ISIT 2011.*