Erasure Codes with Efficient Repair

Ramakrishna Gummadi
Department of Electrical and Computer Engineering,
University of Illinois at Urbana-Champaign.
gummadi2@illinois.edu

R S Sreenivas
Department of Industrial Systems Engineering,
University of Illinois at Urbana-Champaign.
rsree@illinois.edu

***Abstract*—We consider erasure coding in the context of storage applications and propose a metric to capture the complexity of symbol operations associated with dynamically repairing erased code symbols. We then propose a simple construction that has a drastically improved repair complexity compared to standard erasure codes at the expense of a slight increase in the overhead. We also propose a systematic rateless code which has constant encoding/decoding complexity, again at the expense of small overhead parameter. The original systematic Raptor code proposed by Shokrollahi has a linear per symbol encoding complexity.**

I. Introduction

While the benefits of erasure coding for storage are well understood with regard to the savings on the number of storage units for a given level of redundancy, other important aspects that favor replication as opposed to coding, are not as well understood. The most significant of these issues ([1]) is the latency related to recovery from failures. Another related issue that has been pointed out is the large state dependency metric, which refers to the need to contact a potentially large number of servers/storage units for each recovery. An efficient decoding/encoding algorithm does not, by itself, provide an efficient recovery algorithm because decoding efficiency in the context of codes designed for communication have a basic assumption of recovering the entire message block.

Even though the storage gains of coding over replication are extremely large (by a factor that scales with the number of message blocks being coded), storage cost itself could be less important compared to other factors like repair latency. As long as a code delivers the order of magnitude level improvements related to storage over replication, the need for strict information theoretic optimality of storage overhead of the code is less significant compared to the ability to repair destroyed code symbols efficiently.

Another issue where fixed rate code designs face an obstacle unlike replication is related to the flexibility of adding or deleting storage units dynamically ([2]). When a code is designed with a fixed rate beforehand, adding or deleting code symbols no longer keeps the guarantees that were provided for the original design.

II. Repair Complexity

The repair complexity of a code refers to the complexity of operations required to reconstruct subsets of its code symbols using other code symbols. More precisely, let C be an erasure code with its symbols indexed by integers from $[n]$. For some $D \subseteq [n]$ (that is destroyed) and another (disjoint) subset $R \subseteq [n]$ (the recovery set), a repair algorithm then reconstructs the code symbols indexed in D using those from R, when feasible. Typically, one may consider $R = [n]/D$. The repair complexity for a given reconstruction instance is then understood to be the average (over the number of elements in D) number of symbol operations performed by the repair algorithm.

$$\mathcal{R}(D) = \frac{\#\text{ of symbol operations in repairing } D}{|D|} \tag{1}$$

A code on source data divided into k symbols is said to have an overhead δ if one has the guarantee that any $k(1+\delta)$ code symbols are sufficient to reconstruct the source data. When $\delta = 0$, we have an optimal code with respect to its overhead. Loosely speaking, the ease of repair is closely related to the overhead. The closer to optimal the overhead is, the harder it gets to repair the code. This tradeoff has been investigated in prior work extensively for a broadly related problem called the repair bandwidth minimization problem. However, our work has some fundamental differences with this line of work as explained below:

Distinction from the bandwidth minimization problem There has been substantial work on the problem of optimizing the repair bandwidth against the overhead (eg. [3] and the references therein). There are however, two fundamental differences with the notion of repair complexity considered here. (1) In this paper, we consider a code symbol as a single atomic unit of memory which can not be split further. In [3], each node stores data that can be split into subunits and the goal is to optimize the bandwidth across the nodes. In other words, any operational complexity within a node is not a part of the optimization. (2) Our goal of optimizing the number of symbol operations required for reconstruction is more closely aligned to disk access than the bandwidth communicated across nodes itself. For bandwidth optimization ([3]), the codes designed are expected to download miniscule uniform amounts of compressed/coded data from each node over a large number of nodes to save on the final bandwidth, whereas the total number of symbol operations performed prior to communication could be quite expensive due to large number of symbols accessed.

A. *Repair complexity of examples*

1) Parity Code: A parity code of dimension k is $\{v \in \{0,1\}^{k+1} : v_{k+1} = \sum_{i=1}^{k} v_i\}$. Let $D = \{i\}$ for some $i \in [k+1]$ and $R = \{[k+1]/D\}$. The repair is performed by the relation $v_i = \sum_{j \neq i} v_j$, which involves k symbol operations. So the repair complexity is k.

2) Repetition Code: Let the code be $\{v \in \{0,1\}^n : v = [\underbrace{v^* \; v^* \ldots v^*}_{l \text{ times}}]\}$ for some $v^* \in \{0,1\}^k$. Let $D \subseteq [kl]$ such that the number of indices in D which are equivalent modulo k is at most $l-1$. To repair D, we need to perform at most $|D|$ symbol operations in copying the corresponding symbol from R, implying a repair complexity of 1. But the overhead for this code is $\delta = l\frac{k-1}{k} + \frac{1}{k} - 1$, which is arbitrarily bad for large values of l.

3) Rateless Codes - Random Linear Codes, LT Codes, Raptor Codes: Consider a rateless code with dimension k, i.e. the source message consists of k symbols and whose encoding/decoding complexities are on average α and β per symbol respectively and the overhead is δ. For RLC, $\alpha = \beta = \theta(k), \delta = o(k)$; for LT codes, $\alpha = \beta = O(\log k), \delta = o(k)$ and for Raptor Codes, $\alpha = \beta = O(1)$ and $\delta = \epsilon$, a small positive number. Consider a recovery set R with $|R| = k(1+\delta)$. A natural repair algorithm is (1) Decode the source symbols, S from R. (2) Encode the missing code symbols D using S. Under the given assumptions, step 1 incurs βk symbol operations and step 2 incurs $\alpha|D|$ symbol operations. Therefore, the repair complexity for D is:

$$\frac{\beta k + \alpha|D|}{|D|} = \beta\frac{k}{|D|} + \alpha$$

This could be efficient if $|D| \approx k$, but when D is small, even constant encoding/decoding complexities do not provide us with a corresponding efficient repair guarantee.

III. The Augmented LT Code

Let $S = \{s_1, \ldots, s_k\}$ represent the data to be stored, divided into fragments which are also called source symbols. Let $c_1, c_2, \ldots$ represent an LT coded stream generated on S with degree distribution Ω on $[k]$. The augmented LT code is defined as the rateless code formed by adjoining the uncoded source symbols to the LT coded stream, i.e. $\{s_1, \ldots, s_k, c_1, c_2, \ldots\}$.

A. *Repair Algorithm for the Augmented LT Code*

Let D be the set of indices of the code symbols to be repaired and R be a recovery set. Let $R_C = R \bigcap \{c_1, c_2, \ldots\}$ and $R_S = R_C \bigcup S$ (so $R = R_C \bigcup R_S$). Let $|D| = t$ and denote $D_S = D \bigcap S$ and $D_C = D/S = D \bigcap \{c_1, \ldots\}$. Consider the case when $|R_C| \geq k(1+\epsilon)$ where $\epsilon > 0$ and k is large. The repair algorithm for D from R is given below.

1) Since $|R_C| \geq k(1+\epsilon)$, w.h.p. it is feasible to iteratively decode S from R_C. Let $s_{\pi(1)}, s_{\pi(2)}, \ldots s_{\pi(k)}$ be a sequence in which the source symbols could be decoded from R_C. Let $c_{\xi(1)}, \ldots, c_{\xi(k)}$ be the corresponding sequence in which code symbols from R_C are processed from the gross ripple during the decoding process[1]. π and ξ can be easily computed without performing any symbol operations, by just processing the packet headers or the random number seed generator used for the code construction. We have:
$$s_{\pi(i)} = c_{\xi(i)} \oplus \sum_{j \in S_i} s_j \tag{2}$$
where $S_i \subseteq \{s_{\pi(1)}, \ldots, s_{\pi(i-1)}\}$ and $|S_i| = deg(c_{\xi(i)}) - 1$.
2) Repair symbols from D_S in the order of their appearance in π, using Equation 2. To rephrase, recover the symbols in D_S in the sequence $(s_{\pi(i_1)}, s_{\pi(i_2)}, \ldots, s_{\pi(i_t)})$ where $i_1, \ldots, i_t$ are ascending. (This is feasible because for each i, the entire set S_i would have been repaired by the time it is used in Equation 2)
3) Recover D_C by a standard Ω-degree distribution encoder..

Lemma 3.1: The expected repair complexity for arbitrary sets D is at most $(1+\epsilon)\Omega'(1)$ while the overhead δ is at most $1+\epsilon$.

Proof: In appendix. ■

The overhead is suboptimal for the above code, but it achieves a tradeoff between the extremes of bad repair complexity and bad overhead.

IV. Augmented Raptor Codes

Let $\epsilon > 0$ be small. Consider a (high rate) precode with code symbols $S_I = \{s_1, \ldots, s_{k(1+\epsilon)}\}$ such that any subset of k symbols from S_I can recover the message. Let Ω be a degree distribution on $[k(1+\epsilon)]$. Let $c_1, c_2, \ldots$ be a fountain coded stream generated by using Ω on S_I. The augmented Raptor code is defined as the rateless code formed by adjoining S_I to this fountain coded stream, i.e. $\{s_1, \ldots, s_{k(1+\epsilon)}, c_1, c_2, \ldots\}$. The next section is on the overhead properties of this code, which is necessary for establishing the repair complexity claims.

A. *Overhead*

Overhead is the smallest $\delta > 0$ such that an arbitrary set, R of $k(1+\delta)$ code symbols can recover the source symbols. We first briefly recall the recovery constraint obtained by ([8], [9]).

Proposition 4.1: ([8], [9]) A total of rk Ω-coded symbols on k input symbols can be used to decode θk input symbols by iterative decoding if (for large k and w.h.p.)

$$r\Omega'(t) + \log(1-t) \geq 0 \quad \forall \quad t \in (0, \theta)$$

Let $R_C \triangleq R \bigcap \{c_1, c_2, \ldots\}$ and $R_S \triangleq R \bigcap S_I = R/R_C$. Since $|R| = k(1+\delta)$, let $|R_C| = k(1+\delta)\alpha$ and $|R_S| = k(1+\delta)(1-\alpha)$ for some $\alpha \in [0,1]$. We have

$$\begin{aligned}|S_I/R_S| &= k(1+\epsilon) - k(1+\delta)(1-\alpha) \\ &= k(\alpha(1+\delta) - (\delta - \epsilon))\end{aligned}$$

[1]Proposition 6.1 in appendix has more explanation.

Define:

$$\lambda \triangleq \frac{|S_I/R_S|}{|S_I|} \tag{3}$$

$$= \frac{\alpha(1+\delta)-(\delta-\epsilon)}{1+\epsilon} \tag{4}$$

The projected code: Since the code symbols from R_S are essentially decoded to begin with, it is useful to view the code symbols from R_C as being generated by a degree distribution, , $\Psi(x)$, on S_I/R_S. The "projected code" has $k(1+\delta)\alpha$ code symbols with degree distribution Ψ on a set of $|S_I/R_S| = k(\alpha(1+\delta)-(\delta-\epsilon))$ input symbols. Since the degree distribution of the code symbols in R_C with respect to S_I is $\Omega(x)$, the degree distribution projected onto S_I/R_S is given by $\Psi(x) \triangleq \Omega(1-\lambda+\lambda x)$ (see, eg. [7]). Denote

$$f = \frac{\delta}{\epsilon} - 1 \tag{5}$$

and

$$M = \frac{1}{\epsilon} + 1 \tag{6}$$

A sufficient condition for recovery is to decode at least k total symbols from S_I. This translates to a recovery requirement of $k - |R_S| = k(1-(1+\delta)(1-\alpha))$ symbols. A sufficient condition for recovery can be obtained by expressing the recovery requirement and overhead available as fractions of the number of input symbols ($|S_I/R_S|$) for the projected code. These are:

Recovery fraction requirement:

$$\frac{1-(1+\delta)(1-\alpha)}{\alpha(1+\delta)-(\delta-\epsilon)} = 1 - \frac{1}{M\lambda}$$

Code symbols fraction available:

$$\frac{(1+\delta)\alpha}{\alpha(1+\delta)-(\delta-\epsilon)} = 1 + \frac{f}{M\lambda}$$

The recovery constraint (Proposition 4.1) now becomes:

$$\forall \lambda \in (\frac{1}{M}, 1),$$

$$\left(1+\frac{f}{M\lambda}\right)\Psi'(t) + \log(1-t) > 0 \quad \forall \;\; t \in \left(0, 1-\frac{1}{M\lambda}\right) \tag{7}$$

Note that f maps to an overhead $\delta = \frac{f+1}{M-1}$ by Equations (5),(6). To rephrase, we are therefore interested in the smallest $f > 0$, for which we can design Ω satisfying the following constraint:

$$\forall \lambda \in \left(\frac{1}{M}, 1\right),$$

$$\left(\lambda+\frac{f}{M}\right)\Omega'(1-\lambda+\lambda t) + \log(1-t) > 0 \;\; \forall \;\; t \in \left(0, 1-\frac{1}{M\lambda}\right) \tag{8}$$

Let $\Omega(t) = \sum_i p_i t^i$. Since $1-\lambda+\lambda t \le \frac{1}{M} \quad \forall t \in \left(0, 1-\frac{1}{\lambda M}\right)$, it follows that $i(1-\lambda+\lambda t)^i$ is dominated by $M(1-\lambda+\lambda t)^M \quad \forall \;\; i > M$. So it suffices to restrict attention to degree distributions with $p_i = 0 \;\; \forall \;\; i > M$ to search over distributions to minimize f.

Given $\mathcal{P}$, any probability distribution with support on integers less than M and $\lambda \in (\frac{1}{M}, 1)$ and $t \in (0, 1-\frac{1}{M\lambda})$, define:

$$\mathcal{D}(M, \mathcal{P}, \lambda, t) \triangleq \tag{9}$$

$$\frac{M}{M-1}\left(\frac{-\log(1-t)}{\sum_{i=1}^{M} i(1-\lambda+\lambda t)^{i-1} p_i} - \lambda\right) + \frac{1}{M-1} \tag{10}$$

Then, we can achieve an overhead δ that is arbitrarily close to δ_{opt} defined as:

$$\delta_{opt} = \inf_{\substack{M>1 \\ \mathcal{P}}} \sup_{\substack{\lambda \in (\frac{1}{M}, 1) \\ t \in (0, 1-\frac{1}{M\lambda})}} \mathcal{D}(M, \mathcal{P}, \lambda, t) \tag{11}$$

It will also be useful to consider the "overhead profile" for each fixed degree distribution ($\mathcal{P}$) and precode (M), which is the overhead that is necessary as a function of λ. More precisely, let:

$$\delta(\lambda, \mathcal{P}, M) \triangleq \sup_{t \in (0, 1-\frac{1}{M\lambda})} \mathcal{D}(M, \mathcal{P}, \lambda, t) \tag{12}$$

Although the overhead of a given design is captured by $\sup_{\lambda \in (\frac{1}{M}, 1)} \delta(\lambda, \mathcal{P}, M)$, it will be of interest to consider the entire profile as a function of λ if we need to optimize the design with assumptions on λ (which translates to assumptions on the relative composition of the recovery set between S_I and $\{c_1, c_2, \ldots\}$, captured by the parameter $\alpha \in [0, 1]$.).

Theorem 4.1: Let $\mu > 0$ and $D > M$. Let Ω be the same distribution as used in Raptor codes ([10]):

$$\Omega(x) = \frac{1}{\mu+1}\left(\mu x + \sum_{i=1}^{D} \frac{x^i}{i(i-1)} + \frac{x^{D+1}}{D}\right)$$

Then, we have the following upper bound on the overhead profile:

$$\delta(\lambda, \Omega, M) < \frac{1}{M-1} + \frac{M}{M-1}\left(\frac{(1+\mu)\log(M\lambda)}{\mu + \log M} - \lambda\right) \tag{13}$$

Proof: In Appendix. ∎

Figure 1 shows a plot of this bound corresponding to a specific choice of the parameters. We now revisit Equation (11) to numerically optimize the overhead. Consider any M and $\delta^* > 0$. We see that an overhead $\delta = \frac{M}{M-1}\delta^* + \frac{1}{M-1}$ is feasible iff the following LP (with variables $p_i, \;\; i \in [M]$) has a feasible solution for some M.

$$\forall \lambda \in (\frac{1}{M}, 1), t \in (0, 1-\frac{1}{M\lambda}) \tag{14}$$

$$(\delta^* + \lambda)\sum_{i=1}^{M} i(1-\lambda+\lambda t)^{i-1} p_i \ge -\log(1-t) \tag{15}$$

$$\sum_{i=1}^{M} p_i = 1 \;\; \text{and} \;\; p_i \ge 0 \;\; \forall \;\; 1 \le i \le M \tag{16}$$

We can approximate this system by choose a fine grid for the parameters λ, t to obtain an LP that has a finite

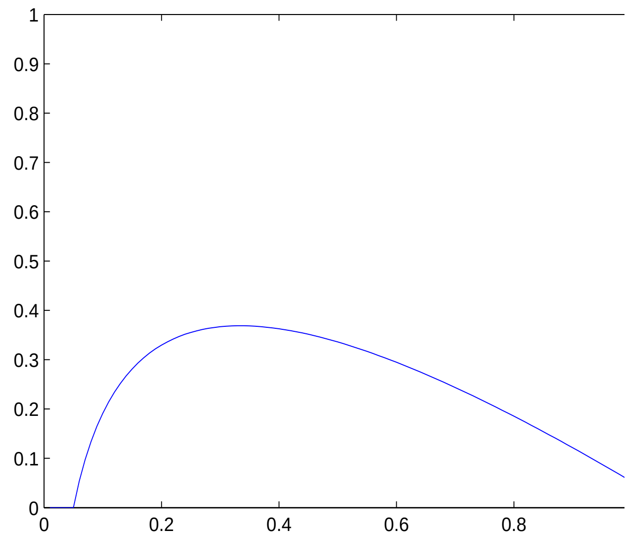

Fig. 1. A plot of the bound on overhead profile in Equation (13) obtained for $M = 20, \mu = 0.001$

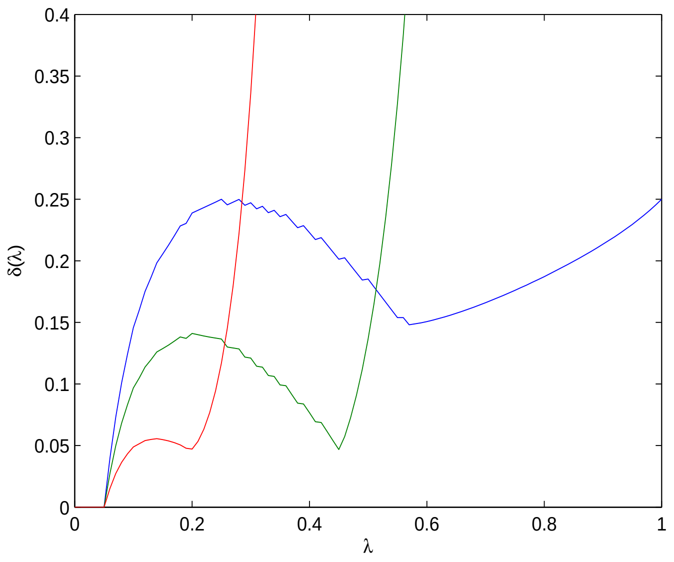

Fig. 2. A plot of three overhead profiles corresponding to three different candidate degree distributions that were obtained by optimizing the average degree over discretized versions of LP constraints (14). The green and red profiles are for distributions that were specifically optimized to minimize the maximum overhead attained for the ranges $\lambda \in (0, 0.5)$ and $\lambda \in (0, 0.2)$ respectively.

number of constraints. This LP can be solved to obtain a candidate degree distribution, whose overhead profile can then be explicitly evaluated numerically using Equation (12) (which is, of course, unconditional on the approximations made while obtaining the candidate degree distribution itself). Given below is a plot of some feasible overhead profiles obtained by optimizing the degree distributions while minimizing the average degree on a support of $M = 20$.

We note from Figure 2 that it is possible to design degree distributions on a support of $M = 20$ that provide an overhead guarantee of at most than $\delta = 0.25$ across all ranges of λ. The reason we plot green and red curves in Figure 2 even though they have ranges of λ which perform worse than the blue curve is to show that it is possible to further optimize the overhead guarantees, if one were to assume restrictions on λ. In Figure 2, the green overhead profile was obtained for a distribution that was optimized for $\lambda \in (0, 0.5)$ wheras the red curve corresponds to $\lambda \in (0, 0.2)$. Recall that λ corresponds to the fraction of symbols in S_I that need to be repaired. If, for example, the designer is confident that the systematic part will never have more than 50% of symbols destroyed, one may use the degree distribution represented by the green curve in its design.

B. Repair Algorithm

Let D be the set of code symbols to be repaired and let R be a recovery set. Let $D_S = D \bigcap S_I$ and $D_C = D \bigcap \{c_1, \ldots\}$. Let $R = R_S \bigcup R_C$ where $R_S = S_I / D_S$ and $R_C = R / R_S$. There are three steps: (1) Repair $D^* \subseteq D_S$ such that $|D_S / D^*| < \epsilon k$ (if $|D_S| \geq k\epsilon$). This can be accomplished using an iterative repair procedure similar to the augmented LT code. (2) Repair D_S / D^* using a repair algorithm for the precode and then (3) Repair D_C using an Ω- degree distribution encoder.

To ensure that Step 1 is feasible, R_C is assumed to be a set of random code symbols (of the smallest size, for repair comlexity claims) that assures the recovery of at least k total symbols from S_I w.h.p. This in turn depends on the size of R_S which determines the λ in equation (4). Let $\delta(\lambda)$ denote the overhead profile for the code used. We have:

$$|R_C| = (1 + \delta(\lambda))\alpha \qquad (17)$$

C. Repair Complexity

To establish the repair complexity, we need to evaluate the number of symbol operations involved in Step 1 while repairing D^*. Let $\delta(\lambda)$ be the overhead profile for the given degree distribution (Section IV-A). The expected number of symbol operations involved in the repair is at most $\Omega'(1)|R_C|$ Therefore, the expected per symbol repair complexity while reconstructing D^* is:

$$\frac{\Omega'(1)|R_C|}{|D^*|} = \frac{k(1 + \delta(\lambda))\alpha}{k(1 - (1 + \delta(\lambda))(1 - \alpha))} \qquad (18)$$

$$= 1 + \frac{\delta(\lambda)}{(1 + \delta(\lambda))\alpha - \delta(\lambda)} \qquad (19)$$

$$= 1 + \frac{M - 1}{M} \frac{\delta(\lambda)}{\lambda - \frac{1}{M}} \text{ using relations } (4), (6) \qquad (20)$$

For example, using the bound at Equation (13) for the Raptor code distribution Ω, we can now establish a constant bound on the repair complexity. For any $\lambda > \frac{1}{M}$, we have the following claim on the derivative of the bound on the overhead profile with respect to λ:

$$\begin{aligned}\delta'(\lambda) &= \frac{M}{M-1}\left(\frac{1+\mu}{(\mu+\log M)\lambda}-1\right)\\ &\le \frac{M}{M-1}\left(\frac{(1+\mu)M}{\mu+\log M}-1\right)\end{aligned}$$

Since $\delta(\frac{1}{M})=0$, this implies that $\forall\lambda\in(\frac{1}{M},1)$ (using Eq (18)) that the per symbol repair complexity for any D^* is at most $\frac{(1+\mu)M}{\mu+\log M}$, which completes the claim. As discussed earlier, if we were to assume a bound for λ's range of interest, we are further able to optimize the repair complexity guarantee using a corresponding optimization in the overhead profile as shown in Figure 2.

V. Systematic Rateless codes with constant per symbol encoding/decoding complexity

The systematic versions of Raptor codes have a bottleneck step in the encoding process, which corresponds to an inverse matrix multiplication (Step 1 of Algorithm 11 in [10]) and consequently has a linear per symbol encoding complexity. The systematic versions of LT codes proposed in [10] avoid this step, but they do not enjoy the constant per symbol encoding/decoding complexity advantage of raptor codes. We note that the augmented Raptor codes can be made systematic rateless codes by choosing a high rate precode that is systematic (eg. from [8]). The encoding/decoding complexities are constant per symbol, although we do sacrifice a constant overhead (of at most 0.25). This could be a useful tradeoff in situations when the reduction in encoding complexity (from linear per symbol to constant per symbol) is attractive in comparison to the additional bandwidth cost due to extra overhead (which can be guaranteed to be at most 25% and can be reduced further if a guarantee can be provided on the fraction of correctly transmitted symbols within the systematic part).

VI. Conclusion

In this paper, we proposed a natural notion of repair complexity for erasure codes with a focus on storage applications and discussed how it differs from the objectives of efficient encoding/decoding. We then designed rateless codes that achieve a significant reduction in the repair complexity from linear per symbol to constant per symbol at the expense of a small overhead inefficiency. The construction also provides us a systematic rateless code that has a constant encoding/decoding complexity at the expense of a small inefficiency in the overhead. Though the original version of systematic Raptor codes is near-MDS, it had a linear per symbol encoding complexity.

Appendix

Proposition 6.1: Consider a collection of k source packets, S and let $I\subseteq S$ denote "side information" (equivalent to assuming that the decoding process has a step 0 in which a genie decodes these packets without using anything). Consider some collection of coded packets formed by XOR'ing subsets of S. Let S^* be the set of new source symbols ($S^*\subseteq S$ and $S^*\bigcap I=\phi$) recovered by the iterative decoding process, with $t=|S^*|$. Thus, there are exactly t steps in the decoding process and at each step, a unique new coded packet is processed from the "gross ripple" and decodes a distinct source packet in S^*. (the ripple terminology has been followed from [5], [6]). Let $(s_{\pi(1)},s_{\pi(2)},\ldots,s_{\pi(t)})$ denote the source packets forming S^* in the sequence in which they are decoded and let $(c_{\xi(1)},\ldots,c_{\xi(t)})$ denote the coded packets were processed in sequence (from the corresponding gross ripple). Since $c_{\xi(i)}$ was processed from the gross ripple to decode $s_{\pi(i)}$ in step i, we must have:

$$c_{\xi(i)} = s_{\pi(i)} \oplus \sum_{j\in S_i} s_j \tag{21}$$

where $S_i \subseteq I\bigcup\{s_{\pi(1)},\ldots,s_{\pi(i-1)}\}$ and $|S_i| = deg(c_{\xi(i)})-1$.

(This is a reformulated version of the idea behind online encoding in [4]).

Proof of Lemma 3.1: Let $|D_S|=t$. The number of symbol operations in restoring $s_{\pi(i)}$ is equal to $deg(c_{\xi(i)})$ (from equation 2). Since the sequence π is uniform over all possibilities when averaged over the randomness of the code, $c_{\xi(i_1)},\ldots,c_{\xi(i_t)}$ has to be a uniform subset from $c_{\xi(1)},\ldots,c_{\xi(k)}$, which are themselves a subset of the set $c_1,\ldots,c_{k(1+\epsilon)}$. Therefore,

$$\begin{aligned}\mathbf{E}\sum_{j=1}^{t}[deg(c_{\xi(i_j)})] &= t\,\mathbf{E}[deg(c_{\xi(i_1)})]\\ &= \frac{t}{k}\sum_{j=1}^{k}\mathbf{E}[deg(c_{\xi(j)})]\\ &\le \frac{t}{k}\sum_{j=1}^{k(1+\epsilon)}\mathbf{E}[deg(c_j)]\\ &= t(1+\epsilon)\Omega'(1)\end{aligned}$$

The number of symbol operations in the rest of the reconstruction is $|D_C|\Omega'(1)$. Therefore, the repair complexity is at most

$$\frac{t(1+\epsilon)\Omega'(1)+|D_C|\Omega'(1)}{t+|D_C|} \le (1+\epsilon)\Omega'(1)$$

The overhead of the augmented LT code is $\delta \le 1+\epsilon$ because any set of $2+\epsilon$ surviving symbols ensures at least $1+\epsilon$ surviving symbols from the set $\{c_1,c_2,\ldots\}$, using which decodability is guaranteed.

■

Proof of Theorem 4.1:

$$\Omega(x) = \frac{1}{\mu+1}\left(\mu x + \sum_{i=1}^{D}\frac{x^i}{i(i-1)} + \frac{x^{D+1}}{D}\right) \tag{22}$$

$$\Omega'(x) = \frac{1}{\mu+1}\left(\mu - \log(1-x) + x^D - \sum_{d=D+1}^{\infty}\frac{x^d}{d}\right)$$

$$\Omega'(1-\lambda+\lambda x) = \frac{1}{\mu+1}\left(\mu - \log\lambda - \log(1-x) + \right.$$
$$\left.(1-\lambda+\lambda x)^D - \sum_{d=D+1}^{\infty}\frac{(1-\lambda+\lambda x)^d}{d}\right)$$

First consider the following lower bound on one of the terms:

$$\begin{aligned}
&(1-\lambda+\lambda x)^D - \sum_{d=D+1}^{\infty}\frac{(1-\lambda+\lambda x)^d}{d}\\
&= (1-\lambda+\lambda x)^D\left(1 - \sum_{D+1}^{\infty}\frac{(1-\lambda+\lambda x)^{d-D}}{d}\right)\\
&\geq (1-\lambda+\lambda x)^D\left(1 - \frac{1}{D+1}\sum_{t=1}^{\infty}(1-\lambda+\lambda x)^t\right)\\
&= (1-\lambda+\lambda x)^D\left(1 - \frac{1-\lambda+\lambda x}{(D+1)\lambda(1-x)}\right)\\
&= (1-\lambda+\lambda x)^D\left(1 + \frac{1}{D+1} - \frac{1}{(D+1)\lambda(1-x)}\right)\\
&\geq (1-\lambda+\lambda x)^D\left(1 - \frac{M}{D}\right) \quad \because x \in \left(0, 1-\frac{1}{M\lambda}\right)
\end{aligned}$$

The sufficient condition becomes

$$\forall\ \ \lambda \in \left(\frac{1}{M}, 1\right), \quad \forall\ \ x \in \left(0, 1-\frac{1}{M\lambda}\right)$$

$$\left(\lambda + \frac{f}{M}\right)\left(\mu - \log\lambda - \log(1-x) + (1-\lambda+\lambda x)^D\left(1-\frac{M}{D}\right)\right)$$
$$> -(1+\mu)\log(1-x)$$

which is equivalent to:

$$\mu + (1-\lambda+\lambda x)^D\left(1-\frac{M}{D}\right) > -\log(1-x)\left(\frac{1+\mu}{\lambda+\frac{f}{M}} - 1\right) - \log\lambda \tag{23}$$

Both LHS and RHS are increasing in x, so this would be implied (although this could be far from being necessary) (by choosing $D \geq M$):

$$\mu > \log(M\lambda)\left(\frac{1+\mu}{\lambda+\frac{f}{M}} - 1\right) - \log\lambda \tag{24}$$

which is equivalent to

$$\mu + \log M > \log(M\lambda)\frac{1+\mu}{\lambda+\frac{f}{M}} \tag{25}$$

which is equivalent to

$$\frac{f}{M} > \frac{(1+\mu)\log(M\lambda)}{\mu+\log M} - \lambda \tag{26}$$

Equivalently, this shows that the Ω specified above already achieves an overhead profile $\delta(\lambda)$ equal to:

$$\delta(\lambda) = \frac{1}{M-1} + \frac{M}{M-1}\left(\frac{(1+\mu)\log(M\lambda)}{\mu+\log M} - \lambda\right) \tag{27}$$

Note that the actual overhead profile achieved will be strictly better than the bound because of a gap between inequalities (23) and (24).

■

References

[1] Z. Zhang, A. Deshpande, X. Ma, E. Thereska, D. Narayanan, *Does erasure coding have a role to play in my data center?*, Microsoft Research Technical Report MSR-TR-2010-52, May 2010. http://research.microsoft.com/apps/pubs/?id=131326

[2] J. Plank, *Erasure Codes for Storage Applications*,4th Usenix Conference on File Storage Technologies, 2005. http://web.eecs.utk.edu/∼plank/plank/papers/FAST-2005.html

[3] A. Dimakis, K. Ramachandran, Y. Wu, C. Suh, *A Survery on Network Codes for Distributed Storage*, on Arxiv, April 2010. http://arxiv.org/abs/1004.4438

[4] R. Gummadi, R. Sreenivas, *Relaying a Fountain Code Across Multiple Nodes*, IEEE ITW 2008.

[5] M. Luby, *LT Codes*, FOCS 2006.

[6] B. Hajek, *Slides from Stochastic Networks Workshop* http://www.ifp.illinois.edu/∼hajek/Papers/HajekStochNets06.pdf

[7] R. Gummadi, A. Shokrollahi, R. Sreenivas, *Broadcasting with Side Information*, IEEE ITW 2010.

[8] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman *Efficient Erasure Correcting Codes*, IEEE Transactions on Information Theory, February, 2001.

[9] R. Darling and J. Norris, *Structure of Large Random Hypergraphs*, Annals of Applied Probability, March, 2005.

[10] A. Shokrollahi, *Raptor Codes*, IEEE Trans on Information Theory, March, 2006.