Simple Regenerating Codes: Network Coding for Cloud Storage

Dimitris S. Papailiopoulos*, Jianqiang Luo†, Alexandros G. Dimakis*, Cheng Huang‡, Jin Li‡
*The University of Texas at Austin
dimitris@utexas.edu, dimakis@austin.utexas.edu
†EMC Data Domain
jianqiang@wayne.edu
‡Microsoft Research, Redmond
cheng.huang@microsoft.com, jinl@microsoft.com

***Abstract*—Network codes designed for distributed storage systems have the potential to provide dramatically higher storage efficiency for better data reliability than replication. One main challenge in the design of such codes is the exact repair problem: if a node storing encoded information fails, in order to maintain the same level of reliability we need to create encoded information at a new node. One of the main open problems in this emerging area has been the design of repair efficient codes that can achieve arbitrarily high data-rates.**

In this work we introduce *simple regenerating codes* (SRCs), a family of codes that are repair efficient and can achieve arbitrarily high data rates. SRCs have very good *repair locality*, which translates to less number of disk accesses during repairs. At the same time, our codes offer very low repair bandwidth during repairs: only a small factor away from the optimal. Our constructions are very simple to implement and perform exact repair by simple XORing of packets.

We experimentally evaluate the proposed codes in a realistic cloud storage simulator and show significant benefits in both performance and reliability compared to replication and standard Reed-Solomon codes.

I. Introduction

Large-scale architectures for storage have traditionally relied on replication for data reliability. However, block replication comes at a tremendous storage cost per block, even for moderate levels of reliability. As the amount of stored data grows faster than hardware infrastructure, this becomes a major design cost.

Erasure codes have been well known to achieve higher reliability with tremendously smaller storage cost [1]. For that reason various erasure codes are currently implemented and deployed in production storage clusters, like Windows Azure [2], the Facebook Analytics Hadoop cluster [3]), Cleversafe, and Wuala.

However, classical erasure codes (such as Reed-Solomon) are highly sub-optimal for distributed settings [4]. For example, the Facebook analytics cluster is currently limiting its Reed-Solomon encoding to a mere 8% of the stored data [3]. Although, this 8% of coded data has substantial projected storage savings, it was reported to generate repair traffic approximately equal to 20% of the total network traffic. This code repair cost is what currently limits a wider code deployment.

Three major repair cost metrics have been studied in the recent literature: *i)* the number of bits communicated in the network, also known as the *repair-bandwidth* [4]–[9], *ii)* the number of bits read during each repair, the *disk-I/O* [7], [10], and *iii)* more recently the number of nodes that participate in the repair process, also known as, *repair locality* [11]–[19]. A major open problem in this literature is the design of codes that can achieve arbitrarily high rates, have favorable repair properties, and are easy to implement.

Our Contribution: In this work, we construct *simple regenerating codes*, codes with good locality and repair bandwidth properties, that are very simple to implement, and can achieve arbitrarily high data rates. Our codes have locality that is only a constant factor away form the optimal bound of [15] and generate repair bandwidth that comes close to the bounds of [4].

An (n, k, f)-SRC is a code that combines f MDS-precodes, has locality $2f$, generates repair bandwidht $\frac{f}{k}$ of the file size, has rate $\frac{f}{f+1}\frac{k}{n}$. The distance of our code is $d = n-k+1$: any k nodes in the system can reconstruct the encoded file. Our constructions are simple to implement, require small finite field size, and perform exact repair by simple XORing of blocks. SRCs can work on top of currently Reed-Solomon encoded blocks: only a few additional XOR parities need to be stored to have the favorable locality and bandwidth properties.

We experimentally evaluate the proposed codes in a realistic cloud storage simulator that models node rebuilds in Hadoop [20]. Our simulator was initially validated on a real system of 16 machines connected by a 1GB/s network. Our subsequent experiment involves 100 machines and compares the performance of SRC to replication and standard Reed-Solomon codes. We find that SRCs add a new attractive point in the design space of redundancy mechanisms for cloud storage.

Most of this work was conducted while the first and the third authors were with the department of Electrical Engineering, at the University of Southern California.

This work was performed while the third author was with Microsoft Research, Redmond, and Wayne State University.

This work was supported, in part, by NSF Career Grant 1055099 and a Research Gift by Microsoft Research.

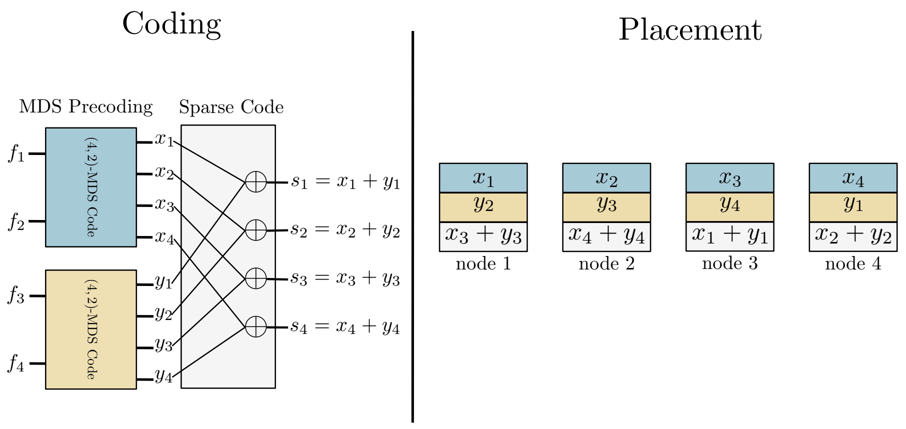


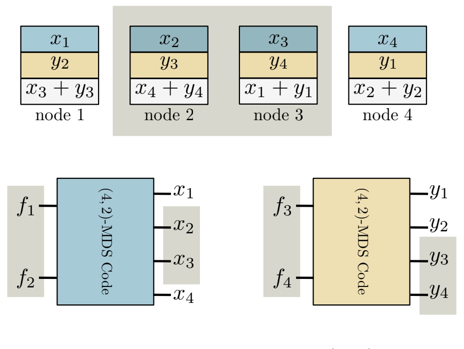
Fig. 1. Example of a $(4, 2)$-SRC. $n = 4$ storage nodes, any 2 disks recover the data and XORs of degree 2 provide simple repair.

II. Simple Regenerating Codes

The first requirement from our storage code is the (n, k) property: a code will be storing information in n storage nodes and should be able *to tolerate any combination of* $n - k$ *failures* without data loss. We refer to codes that have this reliability as "(n, k) *erasure codes*," or codes that have "*the* (n, k) *property*."

One well-known class of erasure codes that have this property is the family of maximum distance separable (MDS) codes [**?**], [**?**]. In short, an MDS code is a way to take a data object of size M, split it into chunks of size M/k and create n coded chunks *of the same size* that have the (n, k) property. It can be seen that MDS codes achieve the (n, k) property with the minimum storage overhead possible: any k storage nodes jointly store M bits of useful information, which is the minimum possible to guarantee recovery.

Our second requirement is efficient exact repair [**?**]. When one node fails or becomes unavailable, the stored information should be easily reconstructable using other surviving nodes. Simple regenerating codes achieve the (n, k) property and simple repair simultaneously by separating the two problems. Two MDS pre-codes are used to provide reliability against any $n-k$ failures, while very simple XORs applied *over the MDS coded packets* provide efficient exact repair when single node failures happen.



x_1 y_2 $x_3 + y_3$ node 1
x_2 y_3 $x_4 + y_4$ node 2
x_3 y_4 $x_1 + y_1$ node 3
x_4 y_1 $x_2 + y_2$ node 4
f_1
f_2
(4, 2)-MDS Code
x_1
x_2
x_3
x_4
f_3
f_4
(4, 2)-MDS Code
y_1
y_2
y_3
y_4

Fig. 2. File reconstruction of a $(4, 2)$-SRC.

We give a first overview of our construction through a simple example in Fig. 1, which shows an $(n = 4, k = 2)$-SRC. The original data object is split in 4 chunks f_1, f_2, f_3, f_4. We first encode $[f_1 \;\; f_2]$ in $[x_1 \;\; x_2 \;\; x_3 \;\; x_4]$ and $[f_3 \;\; f_4]$ in $[y_1 \;\; y_2 \;\; y_3 \;\; y_4]$ using any standard $(4, 2)$ MDS code. This can be easily done by multiplication of the data with the 2×4 generator matrix $\mathbf{G}$ of the MDS code to form $[x_1 \; x_2 \; x_3 \; x_4] = [f_1 \; f_2]\mathbf{G}$ and $[y_1 \; y_2 \; y_3 \; y_4] = [f_3 \; f_4]\mathbf{G}$. Then we generate a parity out of each "level" of coded chunks, i.e., $s_i = x_i + y_i$, which results in an aggregate of 12 coded chunks. We circularly place these coded chunks in 4 nodes, each storing 3, as shown in Fig. 1.

It is easy to check that this code has the (n, k) property and in Fig. 2 we show an example by failing nodes 1 and 4. Any two nodes contain two x_i and two y_i coded chunks which through the outer MDS codes can be used to recover the original data object. We note that the parity chunks are not used in this process, which shows the sub-optimality of our construction.

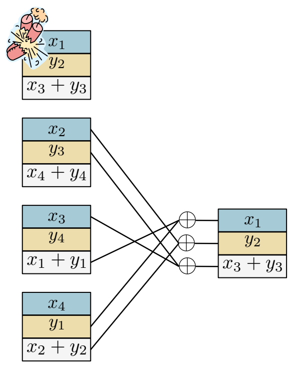


Fig. 3. The repair of node 1 in a $(4, 2)$-SRC

In Fig. 3, we give an example of a single node repair of the $(4, 2)$-SRC. We assume that node 1 is lost and a newcomer joins the system. To reconstruct x_1, the newcomer has to download y_1 and s_1 from nodes 3 and 4. This simple repair scheme is possible due to the way that we placed the coded chunks in the 4 storage nodes: each node stores 3 chunks with different index. The newcomer reconstructs each lost chunk by downloading, accessing, and XORing 2 other coded chunks. In this process the outer MDS codes are not used. In the following subsection, we present the general (n, k) construction.

A. Code Construction

Let a file $\mathbf{f}$, of size $M = 2k$, that we cut into 2 parts $\mathbf{f} = \left[\mathbf{f}^{(1)}\ \mathbf{f}^{(2)}\right]$ where $\mathbf{f}^{(i)} \in \mathbb{F}^{1\times k}$, for $i \in [2]$, $[N] = \{1, \dots, N\}$, and $\mathbb{F}$ is the finite field over which all operations are performed. Our coding process, is a two-step one: first we independently encode each of the file parts using an outer MDS code and generate simple XORs out of them. Then, we place in a specific way the coded chunks and the simple XORs in n storage components. This encode and place scheme enables easy repair of lost coded chunks and arbitrary erasure tolerance.

We use an (n, k) MDS code to encode *independently* each of the 2 file parts $\mathbf{f}^{(1)}$ and $\mathbf{f}^{(2)}$, into two coded vectors, $\mathbf{x}$ and $\mathbf{y}$, of length n

$$\mathbf{x} = \mathbf{f}^{(1)}\mathbf{G} \text{ and } \mathbf{y} = \mathbf{f}^{(2)}\mathbf{G}, \tag{1}$$

where $\mathbf{G} \in \mathbb{F}^{k\times n}$ is the generator matrix of the (n, k) MDS code. The maximum distance of the code ensures that any k encoded chunks of $\mathbf{x}$ or $\mathbf{y}$ can reconstruct $\mathbf{f}^{(1)}$ or $\mathbf{f}^{(2)}$, respectively. Then, we generate a simple XOR vector

$$\mathbf{s} = \mathbf{x} + \mathbf{y}. \tag{2}$$

The above process yields $3n$ chunks: $2n$ coded chunks in the vectors $\mathbf{x}$ and $\mathbf{y}$, and n simple parity chunks in the vector $\mathbf{s}$.

We proceed by placing these $3n$ chunks in n storage nodes in the following way: each storage node will store 3 coded chunks, one from $\mathbf{x}$, one from $\mathbf{y}$, and one from the parity vector $\mathbf{s}$. We require that these 3 coded chunks do not share a subscript. The following circular placement of chunks satisfies this requirement for any $n \geq 2$.

node 1	node 2	...	node $n-2$	node $n-1$	node n
x_1	x_2	...	x_{n-2}	x_{n-1}	x_n
y_2	y_3	...	y_{n-1}	y_n	y_1
s_3	s_4	...	s_n	s_1	s_2

B. Erasure Resiliency and Effective Coding Rate

The (n, k)-SRC can tolerate any possible combination $n-k$ erasures and has effective coding rate $\frac{2}{3}\cdot\frac{k}{n}$. The (n, k) property of the SRC is inherited by the underlying MDS outer codes: we can always retrieve the file by connecting to any subset of k nodes of the storage array. Any subset of k nodes contain k coded chunks of each of the two file parts $\mathbf{f}^{(1)}$ and $\mathbf{f}^{(2)}$, which can be retrieved by inverting the corresponding $k \times k$ submatrices of the MDS generator matrix $\mathbf{G}$.

The coding rate (space efficiency) R of the (n, k)-SRC is equal to the ratio of the total amount of useful stored information, to the total amount of data that is stored

$$R = \frac{\text{file size}}{\text{storage spent}} = \frac{2\cdot k}{3\cdot n}, \tag{3}$$

and is upper bounded by $\frac{2}{3}$. If we fix the erasure tolerance, $n - k = m$, an SRC can have R arbitrarily close to $\frac{2}{3}$ since $\frac{2}{3}\frac{k}{k+m} \xrightarrow{k\to\infty} \frac{2}{3}$.

C. Repairing Lost Chunks

When a single node is lost in an (n, k)-SRC the repair process is initiated. The SRCs enable easy repair of single lost coded chunks, or single node failures, due to the fact that each chunk that is lost shares an index with 2 more coded chunks stored in 2 nodes. By contacting these 2 remaining nodes, we can repair the lost chunk by a simple XOR operation. The repair of a single node failure costs 6 in repair bandwidth and chunk reads, and 4 in disk accesses.

Let for example node $i \in [n]$ fail, that say contains the chunks x_i, y_{i+1}, and s_{i+2}.[1] Then, to reconstruct x_i, we need to connect to node $i-2$ that contains the parity s_i and to node $i-1$ that contains y_i and simply XOR the two chunks. We can similarity repair y_{i+1} by accessing disk $i+1$ and downloading x_{i+1}, then accessing disk $i-1$ and downloading s_{i+1}, and then by XORing the downloaded chunks. Again, the simple parity s_{i+2} is repaired in a similar manner by accessing nodes $i+2$ and $i+1$ from which we download, read, and XOR chunks x_{i+2} and $\mathbf{y}_{i+2}$, respectively. Hence, the repair of a single chunk costs 2 disk accesses, 2 chunk reads, and 2 downloads. To repair a single node failure an aggregate of 6 chunk downloads and reads is required. The set of disks that are accessed to repair all chunks of node i is $\{i-2, i-1, i+1, i+2\}$, for $i \in [n]$, Hence, the number of disk accesses is $d = \min(n-1, 4)$.

III. SRC: The General Construction

In this section we generalize the $f = 2$ construction, to the (n, k, f) SRC. For the general (n, k, f)-SRC, we use f parallel and identical MDS pre-codes and generate a single parity vector from f encoded parts. We circularly place the generated chunks in n storage nodes. The (n, k, f) SRC is an (n, k) erasure code with rate $R = \frac{f}{f+1}\frac{k}{n}$, i.e., the SRC always attains a $\frac{f}{f+1}$ fraction of the space efficiency of an (n, k) MDS code, for the same reliability, but with simple and low cost node repair. We perform single node repairs in the same manner as the $f = 2$ case: to repair a chunk, we access f nodes and perform a simple addition. For any f, the overhead to repair a single chunk is a fraction $\frac{1}{k}$ of tha filesize and the number of chunk reads and disk accesses is f. The repair of a single node failure costs $(f+1)\frac{M}{k}$ in repair bandwidth and we prove that the total number of disk accesses needed for a single node failure is exactly $2\cdot f$. We proceed by introducing the general code construction and showing its properties.

A. Encoding, Erasure Resilience, and Rate

Let a file $\mathbf{f}$, of size $M = fk$, that is subpacketized in f parts,

$$\mathbf{f} = \left[\mathbf{f}^{(1)} \dots \mathbf{f}^{(f)}\right] \in \mathbb{F}_q^{1\times dk}, \tag{4}$$

[1]The index subtractions an additions are performed over the ring $\{1, \dots, n\}$ (for example $1 - 1 = n$).

with each $\mathbf{f}^{(i)}$, $i \in [f]$, having size k. We encode each of the f file parts independently, into vectors $\mathbf{x}^{(i)}$ of length n, for $i \in [f]$, using an (n,k) MDS code. That is, we have

$$\mathbf{x}^{(1)} = \mathbf{f}^{(1)}\mathbf{G}, \ \ldots, \ \mathbf{x}^{(f)} = \mathbf{f}^{(f)}\mathbf{G} \tag{5}$$

where $\mathbf{G} \in \mathbb{F}_q^{k\times n}$ is the MDS generator matrix.

Remark 1: We note that this MDS code can be any scalar or array (n,k) MDS code, since we pose no requirements on its design or finite field size.

We generate a single parity sum vector from all the coded vectors

$$\mathbf{s} = \sum_{i=1}^{f} \mathbf{x}^{(i)}. \tag{6}$$

This process yields in total fn coded chunks in the $\mathbf{x}^{(i)}$ vectors and n parity chunks in $\mathbf{s}$, i.e., we have an aggregate of $(f+1)n$ chunks available to place in n nodes.

We will circlarly place these $(f+1)n$ chunks in n storage nodes, with each node storing f coded chunks and 1 parity sum chunk, hence

$$\alpha_{\text{SRC}} = f + 1 = \frac{f+1}{f}\frac{M}{k}. \tag{7}$$

The placement should again obey the property that enables easy repair: no two chunks within a storage node should share the same subscript. To ensure successful repair we also require that $f \leq n$. Below we state the circular placement of chunks in the i-th node, for $i \in [n]$

$$\begin{bmatrix} x_i^{(1)} \\ x_{i\oplus 1}^{(2)} \\ \vdots \\ x_{i\oplus(f-1)}^{(f)} \\ s_{i\oplus f} \end{bmatrix}, \tag{8}$$

which results in the following array of n storage nodes

node 1	node 2	$\ldots$	node $n-1$	node n
$x_1^{(1)}$	$x_2^{(1)}$	$\ldots$	$x_{n-1}^{(1)}$	$x_n^{(1)}$
$x_2^{(2)}$	$x_3^{(2)}$	$\ldots$	$x_n^{(2)}$	$x_1^{(2)}$
$x_3^{(3)}$	$x_4^{(3)}$	$\ldots$	$x_1^{(3)}$	$x_2^{(3)}$
$\vdots$	$\vdots$	$\ldots$	$\vdots$	$\vdots$
$x_f^{(f)}$	$x_{f\oplus 1}^{(f)}$	$\ldots$	$x_{f\oplus(n-2)}^{(f)}$	$x_{f\oplus(n-1)}^{(f)}$
$s_{f\oplus 1}$	$s_{f\oplus 2}$	$\ldots$	$s_{f\oplus(n-1)}$	$s_{f\oplus n}$

.

Then, we have the following theorem.

Theorem 1: The (n,k,f)-SRC can tolerate any combination of $n-k$ node erasures and has coding rate $\frac{f}{f+1}\cdot\frac{k}{n}$.

The f MDS pre-codes guarantee perfect file reconstruction posterior to any $n-k$ erasures. The file can always be reconstructed by connecting to any k nodes: any collection of k nodes contain fk distinct coded chunks, k of each file part. Each of these k-tuples can give back the information chunks of a single file part due to the f identical MDS pre-codes, and eventually we can reconstruct the entire file by merging all k file parts.

Let for example a subset of k nodes with index sets

$$\mathcal{N}(j) = \{i_1 + (j-1), \ldots, i_k + (j-1)\} \tag{9}$$

for $j \in [f]$. The storage nodes indexed by the set $\mathcal{N}(1)$ contain the coded chunks $\mathbf{x}_{\mathcal{N}(j)}^{(j)}$, for all $j \in [f]$. We can reconstruct the file parts $\mathbf{f}^{(j)}$ by inverting the corresponding $k\times k$ submatrices of the MDS generator, i.e.,

$$\mathbf{f}^{(j)} = \mathbf{x}_{\mathcal{N}(j)}^{(j)}\mathbf{G}_{\mathcal{N}(j)}^{-1}, \tag{10}$$

for $j \in \{0, \ldots, f-1\}$. Hence the SRC is an (n,k) erasure code.

The effective coding rate of the (n,k,f) SRC is equal to the ratio of the initial filesize to the expedited storage

$$R_{\text{SRC}} = \frac{\text{filesize}}{\text{storage spent}} = \frac{f\cdot k}{(f+1)\cdot n}, \tag{11}$$

and it is a fraction $\frac{f}{f+1}$ of the coding rate of an (n,k) MDS code, hence is upper bounded by

$$\frac{f}{f+1}\frac{k}{k+m} \xrightarrow{k\to\infty} \frac{f}{f+1}. \tag{12}$$

In Fig. 3, we show the effective coding rate of a $(20,17,f)$ SRC code for $f \in [10]$, compared with the coding rate of a $(20,17)$ MDS code. Both codes can tolerate 3 failures. We see how the rate of our code increases with f and approaches that of the MDS code.

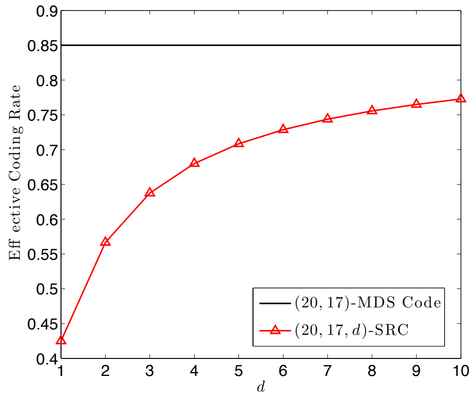


Fig. 4. Rate comparison of a $(20,17,f)$-SRC and a $(20,17)$-MDS Code.

B. Repairing Lost Elements

In this subsection we prove the repair properties of the SRC, which are summarized in the following theorem.

Theorem 2: The repair of a single chunk of the (n,k,f)-SRC, where each node stores $\alpha_{\text{SRC}} = \frac{f+1}{f}\cdot\frac{M}{k}$, costs $\frac{M}{k}$ in repair bandwidth and f in chunk reads, and disk accesses. The repair of a single node failure costs

$$\gamma_{\text{SRC}} = (f+1)\frac{M}{k} \tag{13}$$

in repair bandwidth, $f(f+1) = (f+1)\frac{M}{k}$ in chunk reads, and

$$d_{\text{SRC}} = \min(2f, n-1) \tag{14}$$

	(n,k)-MDS	$(n,k,d=n-1)$-MSR	$(n,k,d=k)$-MBR	$(n,k,d=n-1)$-MBR	(n,k,f)-SRC
Storage per node (α)	M/k	M/k	$2\frac{k}{k+1}M/k$	$\frac{2(n-1)}{2(n-1)-k+1}M/k$	$\frac{f+1}{f}M/k$
Repair Bandwidth (γ)	M	$\frac{n-1}{n-k}M/k$	$2\frac{k}{k+1}M/k$	$\frac{2(n-1)}{2(n-1)-k+1}M/k$	$(f+1)M/k$
Disk Accesses (d)	k	$n-1$	k	$n-1$	$2\cdot f$
Rate (R)	k/n	k/n	$\frac{1}{2}\cdot\frac{k+1}{n}\leq\frac{1}{2}$	$\leq\frac{1}{2}$	$\frac{f}{f+1}k/n$

Fig. 5. (n,k,f)-SRC Performance

in disk accesses.

To prove the above, we let node $i\in[n]$ fail. A newcomer node can reconstruct the lost chunk $x_{i\oplus(l-1)}^{(l)}$ by accessing all f nodes in the set

$$\mathcal{S}_i(l)=\{i\ominus(f-1+l),i\ominus(f-2+l),\ldots,i\ominus l\}\backslash i. \quad (15)$$

and downloading the chunk of each node that has the same subscript $i\oplus(l-1)$ as the lost chunk. For example to reconstruct $x_i^{(1)}$ we need to perform the following steps:

Step	Repair chunk $x_i^{(1)}$:
1	`Access Disk` $i\ominus 1$ `and download` $x_i^{(2)}$
2	`Access Disk` $i\ominus 2$ `and download` $x_i^{(3)}$
$\vdots$	$\vdots$
f-1	`Access Disk` $i\ominus(f-1)$ `and download` $x_i^{(f)}$
f	`Access Disk` $i\ominus f$ `and download` s_i
f+1	`restore` $x_i:=s_i-\sum_{l=2}^{f}x_i^{(l)}$

Hence, repairing a single coded chunk requires $f=\frac{M}{k}$ chunk downloads reads, and disk accesses. To reconstruct the parity sum chunk $s_{i\oplus f}$, we need to connect to the f nodes that contain the chunks $x_{i\oplus f}^{(l)}$, $l\in[f]$ which generate it.

To repair a single node failure we need to communicate and read $(f+1)f=(f+1)\frac{M}{k}$ symbols. The total number of disk accesses is given by the number of distinct indices in the set

$$\mathcal{S}_i=\bigcup_{l=1}^{f+1}\mathcal{S}_i(l). \quad (16)$$

To enumerate the distinct indeces in $\mathcal{S}_i$, we first count the number of distinct indeces between sets $\mathcal{S}_i(l)$ and $\mathcal{S}_i(l+1)$ for all $l\in[f]$. We observe that

$$\mathcal{S}_i(l)\cup\mathcal{S}_i(l+1)=\{i\ominus(f-1+l),i\ominus(f-2+l),\ldots,i\ominus(l-1)\}\backslash i$$

and

$$|\mathcal{S}_i(l)\cup\mathcal{S}_i(l+1)|=f+1, \quad (17)$$

that is, for any two "consecutive" chunk repairs, we need to access $f+1$ storage nodes. Starting with f disk accesses for the first chunk repair, each additional chunk repair requires an additional disk access, with respect to what has already been accessed. The total number of disks accessed is

$$\begin{aligned} d_{\text{SRC}} &= (\text{\# of disks accesses for chunk } x_i^{(1)}) \\ &\quad + (\text{\# of disks accesses for chunk } x_{i\oplus1}^{(2)}) \\ &\quad \vdots \\ &\quad + (\text{\# of disks accesses for chunk } s_{i\oplus f}) \\ &= f+\underbrace{1+1+\ldots+1}_{f\text{ additional disks accesses}}=2\cdot f \end{aligned} \quad (18)$$

Therefore, to repair a single node failure an aggregate of $2f$ disk accesses is required, when $2f\leq n-1$. If $2f<n-1$ then the number of total disk accesses is $n-1$.

In Fig. 5, we give a comparison table between MDS, MSR, MBR, and Simple Regenerating Codes, with respect to 1) storage capacity per node α, 2) repair bandwidth per single node repair γ, 3) number of disk accesses per single node repair d, and 4) effective coding rate R. We consider MSR and MBR codes that connect to $d=\{k,n-1\}$ remaining nodes for a single node failure.

Remark 2: Regenerating Codes [?] have the property that a single node failure can be repaired by any subset of d remaining nodes, and $k\leq d\leq n-1$ is fixed by the specific code design. In sharp contrast, SRCs are look-up repair codes: for a single node failure, only a specific d_{SRC} subset of the remaining nodes can reconstruct the file and d_{SRC} can be a constant, or a function of k that potentially grows much slower than $\Theta(k)$.

C. Asymptotics of the SRC and links to MDS codes

In this subsection, we consider the asymptotics of the SRC. What happens if we fix $R=\frac{k}{n}$ and let the degree of parities f grow as a function of k? Let for example

$$f=\log(k). \quad (19)$$

Then, the repair of a single node costs $\gamma_{\text{SRC}}=(\log(k)+1)M/k$, with $d_{\text{SRC}}=2f=2\log(k)$. In comparison, a single node failure of an (n,k) MSR code costs $\gamma_{\text{MSR}}=\frac{n-1}{n-k}M/k$. If we let k and n grow and fix $R=\frac{k}{n}$ we obtain

$$\frac{\gamma_{\text{SRC}}}{\gamma_{\text{MSR}}}=\frac{\log(k)+1}{\frac{n-1}{n-k}}=\frac{\log(k)+1}{\frac{1/Rk-1}{(1/R-1)k}}\longrightarrow\log(k). \quad (20)$$

Moreover, the effective coding rate of the SRC is given by

$$\frac{f}{f+1}\frac{k}{n}=\frac{\log(k)}{\log(k)+1}\frac{k}{n}\overset{k\to\infty}{\longrightarrow}R. \quad (21)$$

Hence, SRCs for $f = \log(k)$ are log-factor suboptimal in repair communication with respect to MSR codes, but are asymptotically optimal in storage, i.e., are assymptotically MDS. Moreover, instead of connecting to all $n-1$ remaining nodes, a single node failure requires $2\log(k)$ disk accesses. Hence, the by sacrificing assymptotically negligible coding rate and a logarithmic fraction of the optimal repair bandwidth, SRCs attain very easy repair with much lower disk accesses.

IV. The Ring Code, a simple SRC based on replication

An interesting case of the SRC is the $f = 1$ case, which we call the (n, k) RING. The, (n, k) RING is constructed after encoding a file of size k using an (n, k) MDS code, wich outputs n encoded elements $x = [x_1, \dots, x_n]$ which we replciate in n nodes. The data replication of the $f = 1$ case comes from the fact that the parity sum vector $\mathbf{s}$ is simply equal to $\mathbf{x}$, since there is only one bunch of coded elements. The encoded storage array of the n nodes is given below, where each node stores $\alpha_{\text{RING}} = 2$ coded elements

node 1	node 2	...	node $n-1$	node n
$x_1^{(1)}$	$x_2^{(1)}$	...	$x_{n-1}^{(1)}$	$x_n^{(1)}$
$x_2^{(2)}$	$x_3^{(2)}$	...	$x_n^{(2)}$	$x_1^{(1)}$

Through this placement each "consecutive" 2 nodes, share a single coded element. In Fig. 5, we give an example of a RING for $n = 5$, where the links between nodes denote the element that is shared among the two connected nodes.

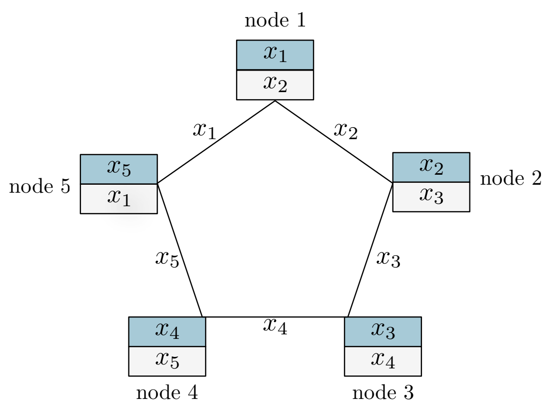


Fig. 6. An $n = 5$ RING

The RING has the simplest repair of all the SRCs, while being able to tolerate $n-k$ failures.. It shares the simple repair process of replications schemes, in which a single lost node is simply read from another node and restored in a newcomer node. Moreover, it enjoys the favorable erasure resiliency of MDS codes, using the underlying MDS code in the same way of the general SRC.

The repair of a single coded element costs 1 in element downloads, element reads, and disk accesses. The repair of a single failed node costs $\gamma_{\text{RING}} = 2\frac{M}{k} = 2$ in element downloads, 2 in element reads, and $d_{\text{RING}} = 2$ in disk accesses. Hence, for each single node failure, only the size of what was lost is accessed and downloaded, which is the theoretic minimum. In that sense, the RING resembles an MBR Code, such as the repair by transfer codes of [?], or the fractional repetition MBR codes of [cite].

V. Simulations

A. Simulator Introduction

We first present the architecture of the cloud storage system that our simulator is modeling. The architecture contains one master server and a number of data storage servers, similar to that of GFS [?] and Hadoop [?]. As a cloud storage system may store up to tens of petabytes of data, we expect numerous failures and hence fault tolerance and high availability are critical. To offer high data reliability, the master server needs to monitor the health status of each storage server and detect failures promptly.

In the systems of interest, data is partitioned and stored as a number of fixed-size chunks, which in Hadoop can be 64MB or 128MB. Chunks form the smallest accessible data units and in our system are set to be 64MB. To tolerate storage server failures, replication or erasure codes are employed to generate redundant coded chunks. Then, several coded chunks are grouped and form a redundancy set [?]. If one chunk is lost, it can be reconstructed from other surviving coded chunks. To repair the chunks due to a failure event, the master server will initiate the repair process and schedule repair jobs.

We implemented a discrete-event simulator of a cloud storage system using a similar architecture and data repair mechanism as Hadoop. To provide accurate simulation results, our simulator models most entities of the involved components such as machines and chunks. When performing repair jobs, the simulator keeps track of the details of each repair process which gives us a detailed performance analysis.

B. Simulator Validation

We first calibrated our simulator to accurately model the data repair behavior of Hadoop. During the validation, we ran one experiment on a real Hadoop system. This system contains 16 machines, which are connected by a 1Gb/s network. Each machine has about 410GB data, namely approximately 6400 coded chunks. Then, we manually failed one machine, and let Hadoop repair the lost data. After the repair was completed, we analyzed the log file of Hadoop and derived the repair time of each chunk. Next, we ran a similar experiment in our simulator. We also collected the repair time of each chunk from the simulation. In Fig. 7, we present the CDF of the repair time of both experiments.

Fig. 7 shows that the repair result of the simulation matches the results of the real Hadoop system very well, particularly when the percentile is below 95. This indicates that the simulator can precisely simulate the data repair process of Hadoop.

C. Storage Cost Analysis

Now we observe how storage overhead varies when we grow (n, k). We compare three codes: 3-way replication, Reed-Solomon (RS) codes, and SRC. To make the storage overhead

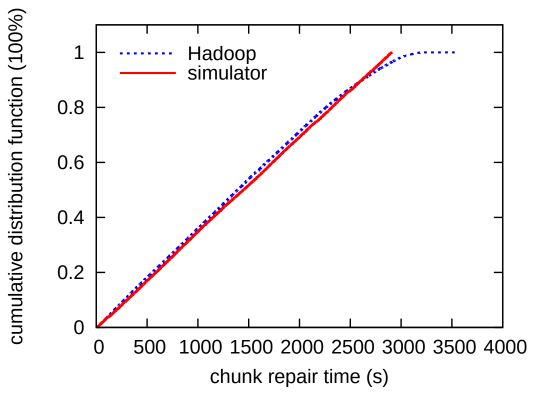


Fig. 7. CDF of repair time

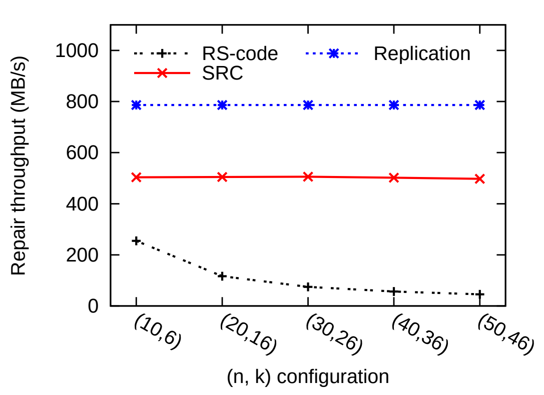


Fig. 9. Repair performance comparison

easily understood, we define the cost of storing one byte as the metric of how many bytes are stored for each useful byte. Obviously, high cost results in high storage overhead. As 3-way replication is a popularly used approach, we use it as the base line for comparison. The results are presented in Fig. 8.

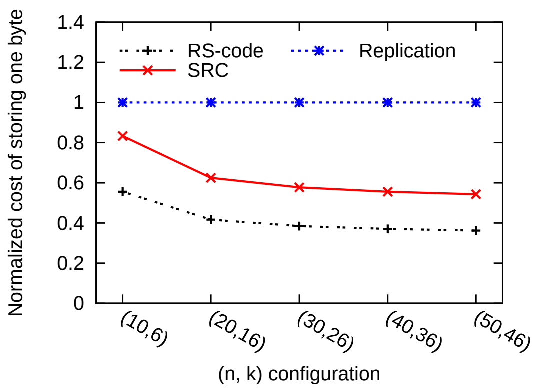


Fig. 8. Storage cost comparison

Fig. 8 shows that when $n - k$ is fixed, the normalized cost of both the RS-code and the SRC decreases as n grows. When (n, k) grows to $(50, 46)$, the normalized cost of the SRC is 0.54, and that of the RS-code is 0.36. In other words, $(50, 46, 2)$ SRCs need approximately half the storage of 3-way replication.

D. Repair Performance

In this experiment, we measure the throughput of repairing one failed data server. The experiment involves a total of 100 machines, each storing 410GB of data. We fail at random one machine and start the data repair process. After the repair is finished, we measure the elapsed time and calculate the repair throughput. The results are shown in Fig. 9. Note that the throughput of using 3-way replication is constant across different (n, k) since there is no such dependency on these parameters.

From Fig. 9 we can make two observations. First, 3-way replication has the best repair performance followed by SRC, while the RS-code offers the worst performance. This is not surprising due to the amount of data that has to be accessed for the repair. Second, the repair performance of the SRC remains constant on various (n, k), but the performance of the RS-code becomes much worse as n grows. This is one of the major benefits of SRCs, i.e., the repair performance can be independent from (n, k). Furthermore, the repair throughput of the SRC is about 500MB/s, approximately 64% of the 3-way replication's performance.

E. Degraded Read Performance

In a real system, repair can take place in two situations. One situation is when we need to repair a failed data storage server. Another situation is when we wish to read a piece of data, but it is stored in a storage server that is currently unavailable. The two situations differ in whether the repaired data is stored or not. The first situation is a regular repair operation, which writes the repaired data back to the system. The second situation repairs the data in the main memory and then simply drops it after serving the read request. We call the latter degraded read. The degraded read performance is important, since clients can notice performance degradations when servers have temporary or permanent failures.

We use a similar experimental environment to what we presented in section V-D. The only difference is after a chunk is repaired, we do not write it back. The performance results are presented in Fig. 10.

We can also make two observations from Fig. 10. First, for all three codes, the performance trend of degraded read performance is similar to that of repair performance, shown in Fig. 9. Second, for a code with the same (n, k), the degraded read performance is higher than that of repair performance, due to less accessed data. Again, SRC achieves approximately 60% degraded read performance of 3-way replication.

F. Data Reliability Analysis

Now we analyze the data reliability of an SRC cloud storage system. We use a simple Markov model [?] to estimate the reliability. For simplicity, failures happen only to disks and we assume no failure correlations. We note that we expect correlated failures to further benefit SRCs over replication since they spread the data to more nodes and hence achieve

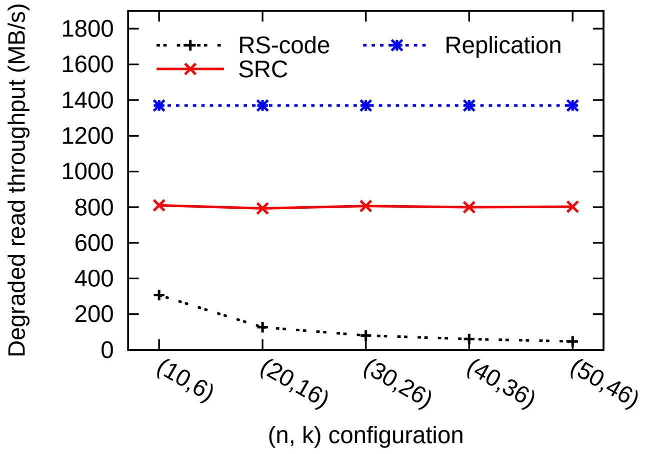


Fig. 10. Degraded read performance comparison

better diversity protection under correlated failure scenarios. This, however, remains to be verified in a more thorough experimental study of coded cloud storage systems.

We assume that the mean time to failure (MTTF) of a disk is 5 years and the system stores 1PB data. To be conservative, the repair time is 15 minutes when using 3-way replication and 30 minutes for the SRC, which is in accordance to Fig. 9. In the case of the RS-code, the repair time depends on k of (n, k). With these parameters, we first measure the reliability of one redundancy set, and then use it to derive the reliability of the entire system. The estimated MTTF of the entire storage system is presented in Fig. 11.

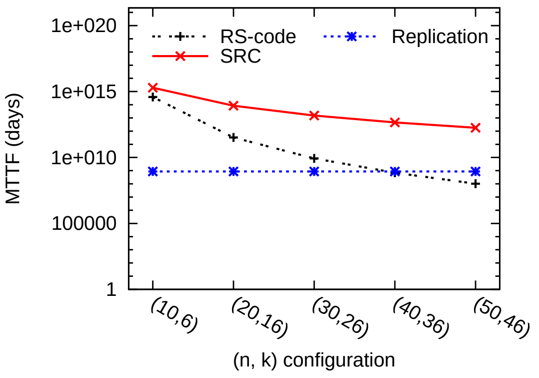


Fig. 11. MTTF comparison

Fig. 11 shows that the data reliability of the 3-way replication is in the order of 10^9. This is consistent with the results in [**?**]. We can observe that the reliability of SRCs is much higher than 3-way replication. Even for the high rate (low storage overhead) $(50, 46)$ case, SRCs are several orders of magnitude more reliable than 3-way replication. This is benefited from the high repair speed of SRCs. RS codes show a significantly different trend. Although the reliability of $(10, 6)$ and $(20, 16)$ are higher than 3-way replication, the reliability of the RS-code reduces greatly when (n, k) grows. This happens because their repair performance rapidly decreases as k grows.

VI. Conclusions

We introduced a novel family of distributed storage codes that are formed by combining MDS codes and simple locally repairable parities for efficient repair and high fault tolerance. One very significant benefit is that the number of nodes that need to be contacted for repair is always 4, that is independent of n, k. Further, SRCs can be easily implemented by combining any prior MDS code implementation with XORing of coded chunks and the appropriate chunk placement into nodes. We presented a comparison of the proposed codes with replication and Reed-Solomon codes using a cloud storage simulator. The main strength of the SRC in this comparison is that it provides approximately four more zeros of data reliability compared to replication, for approximately half the storage. The comparison with Reed-Solomon leads almost certainly to a win of SRCs in terms of repair performance and data availability when more storage is allowed. Our preliminary investigation therefore suggests that SRCs should be attractive for real cloud storage systems. In conclusion, we think that SRCs add new feasible points in the tradeoff space of distributed storage codes.

References

[1] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," *Peer-to-Peer Systems*, pp. 328–337, 2002.
[2] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *USENIX Annual Technical Conference (USENIX ATC)*, 2012.
[3] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: Novel erasure codes for big data," *Proceedings of the VLDB Endowment (to appear)*, 2013.
[4] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Information Theory, IEEE Transactions on*, vol. 56, no. 9, pp. 4539–4551, 2010.
[5] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction," *Information Theory, IEEE Transactions on*, vol. 57, no. 8, pp. 5227–5239, 2011.
[6] C. Suh and K. Ramchandran, "Exact-repair mds code construction using interference alignment," *Information Theory, IEEE Transactions on*, vol. 57, no. 3, pp. 1425–1442, 2011.
[7] I. Tamo, Z. Wang, and J. Bruck, "Mds array codes with optimal rebuilding," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pp. 1240–1244, IEEE, 2011.
[8] V. R. Cadambe, C. Huang, S. A. Jafar, and J. Li, "Optimal repair of mds codes in distributed storage via subspace interference alignment," *arXiv preprint arXiv:1106.1250*, 2011.
[9] D. S. Papailiopoulos, A. G. Dimakis, and V. R. Cadambe, "Repair optimal erasure codes through hadamard designs," in *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pp. 1382–1389, IEEE, 2011.
[10] O. Khan, R. Burns, J. Plank, and C. Huang, "In search of i/o-optimal recovery from disk failures," in *Proceedings of the 3rd USENIX conference on Hot topics in storage and file systems*, pp. 6–6, USENIX Association, 2011.
[11] J. Han and L. A. Lastras-Montano, "Reliable memories with subline accesses," in *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pp. 2531–2535, IEEE, 2007.
[12] C. Huang, M. Chen, and J. Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," in *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*, pp. 79–86, IEEE, 2007.

[13] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, “On the locality of codeword symbols,” *Information Theory, IEEE Transactions on*, vol. 58, no. 11, pp. 6925–6934, 2011.

[14] F. Oggier and A. Datta, “Self-repairing homomorphic codes for distributed storage systems,” in *INFOCOM, 2011 Proceedings IEEE*, pp. 1215–1223, IEEE, 2011.

[15] D. S. Papailiopoulos and A. G. Dimakis, “Locally repairable codes,” in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pp. 2771–2775, IEEE, 2012.

[16] N. Prakash, G. M. Kamath, V. Lalitha, and P. V. Kumar, “Optimal linear codes with a local-error-correction property,” in *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*, pp. 2776–2780, IEEE, 2012.

[17] G. M. Kamath, N. Prakash, V. Lalitha, and P. V. Kumar, “Codes with local regeneration,” *arXiv preprint arXiv:1211.1932*, 2012.

[18] A. Rawat, O. Koyluoglu, N. Silberstein, and S. Vishwanath, “Optimal locally repairable and secure codes for distributed storage systems,” *arXiv preprint arXiv:1210.6954*, 2012.

[19] I. Tamo, D. S. Papailiopoulos, and A. G. Dimakis, “Optimal locally repairable codes and connections to matroid theory,” *arXiv preprint arXiv:1301.7693*, 2013.

[20] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pp. 1–10, IEEE, 2010.