Beehive: Erasure Codes for Fixing Multiple Failures in Distributed Storage Systems

Jun Li, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto, Canada

Abstract

Distributed storage systems have been increasingly deploying erasure codes (such as Reed-Solomon codes) for fault tolerance. Though Reed-Solomon codes require much less storage space than replication, a significant amount of network transfer and disk I/O will be imposed when fixing unavailable data by reconstruction. Traditionally, it is expected that unavailable data are fixed separately. However, since it is observed that failures in the data center are correlated, fixing unavailable data of multiple failures is both unavoidable and even common. In this paper, we show that reconstructing data of multiple failures in batches can cost significantly less network transfer and disk I/O than fixing them separately. We present *Beehive*, a new design of erasure codes, that can fix unavailable data of multiple failures in batches while consuming the optimal network transfer with nearly optimal storage overhead. Evaluation results show that Beehive codes can save network transfer by up to 69.4% and disk I/O by 75% during reconstruction.

1 Introduction

Large-scale distributed storage systems, especially ones in data centers, store a massive amount of data that are also increasing rapidly. Running upon commodity hardware, these storage systems are expected to keep data available, against software and hardware failures that make data unavailable on a daily basis [12]. Traditionally, replicated data are employed by distributed storage systems to keep data available. For example, three replicas are stored by default in the Hadoop Distributed File System (HDFS) [2].

However, storing multiple copies of the original data brings expensive overhead to the storage system. For example, three copies mean that only 33% of the total storage space can be effectively used. Therefore, distributed storage systems (*e.g.,* [6]) have been replacing replication with erasure codes, especially for cold or archival storage. By migrating from replication to erasure codes, distributed storage systems can enjoy a better capability to tolerate unavailable data and meanwhile save storage overhead. Among erasure codes, Reed-Solomon (RS) codes become the most popular choice as RS codes make the optimal usage of storage space while providing the same level of fault tolerance.

To achieve fault tolerance with RS codes, we need to assume that data are stored in *blocks* with a fixed size, a common practice for most distributed storage systems. With k data blocks, RS codes compute r parity blocks, such that any k of the total $k+r$ blocks can recover all data blocks. The data blocks and their corresponding parity blocks belong to the same *stripe*. Therefore, such RS codes can tolerate at most r failures within the same stripe. For example, RS codes with $k = 4$ and $r = 2$ can tolerate any two missing blocks with $1.5x$ storage overhead, while three-way replication, achieving the same level of fault tolerance, requires $3x$ storage overhead.

Once one block becomes unavailable, the missing data should be fixed through a *reconstruction* operation and saved at some other server. Under RS codes, the reconstruction operation requires to download k existing blocks and then decode the missing block, imposing k times of the network transfer under replication. It has been reported from a Facebook's cluster that reconstructing unavailable data under RS codes can increase more than 100 TB of data transfer in just one day [10]. Besides, the same amount of disk I/O will also be imposed on the servers that store the downloaded blocks.

To save network transfer during reconstruction, there have been considerable interests in the construction of a class of erasure codes called *minimum-storage regenerating* (MSR) codes (*e.g.,* [11]). The same as RS codes, MSR codes also make the optimal usage of storage space. However, MSR codes can significantly save network transfer during reconstruction. As shown in Fig. 1a, we assume that the size of each block is 128 MB and

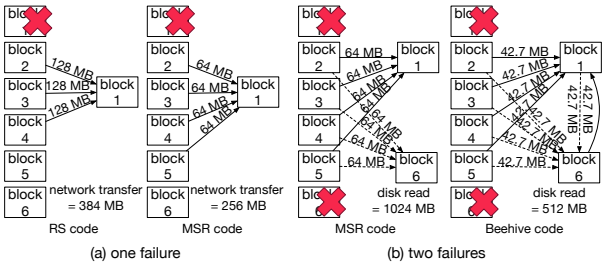

Figure 1: The network transfer and disk read imposed by the reconstruction under RS, MSR and Beehive codes, with $k = 3$, $r = 3$, and blocks of size 128 MB.

let $k = 3$ and $r = 3$ for both RS codes and MSR codes here. While RS codes need to download three blocks to reconstruct one missing block, MSR codes only need to download a fraction of each block. In this example, though MSR codes require to download data from one more block, the total network transfer is still saved by 33.3% as only a half of network transfer is imposed on each block. Nevertheless, even though only a fraction of a block is required for the reconstruction, in general it has to be encoded from the whole block.[1] Therefore, MSR codes do not ease but further increase the overhead of disk I/O, as the reconstruction requires to download data from even more blocks than RS codes.

Traditionally, it is assumed that when there are unavailable blocks, distributed storage systems will reconstruct them separately. However, inside data centers, data unavailability events can be correlated. For example, many disks fail at similar ages [8]. When one disk fails, it suggests a high probability that some other disks fail roughly at the same time. Not just limited to disk failures, correlated data failures can also happen due to various reasons [5] such as switch failures, power outages, maintenance operations, or software glitches. Taking advantages of the correlated failures, we investigate the construction of erasure codes that allows us to reconstruct multiple missing blocks in batches, to save both network transfer and disk I/O during reconstruction.

In this paper, we propose a new family of erasure codes, called *Beehive*, that reconstruct multiple blocks at the same time. An instant benefit this brings is that each block will only be read once to reconstruct multiple blocks. As illustrated in Fig. 1b, the total amount of disk read is saved by 50% when we reconstruct two blocks together at the same time, while we can even further save network transfer as well. In fact, Beehive codes achieve the optimal network transfer per block in the reconstruction operation. The construction of Beehive codes is built on top of MSR codes. We implement Beehive codes in C++ and evaluate the performance of Beehive on Amazon EC2. The experiment results show that compared to MSR codes, Beehive codes can save up to 42.9% of network traffic (with more savings under RS codes) and up to 75% of disk I/O during reconstruction. Though Beehive codes store less actual data than RS or MSR codes with the same storage space, we show that this overhead is marginal.

[1] There exist some constructions [9, 14] of erasure codes that support to obtain a fraction of block without any encoding operations. However, typically MSR codes do not optimize for disk I/O.

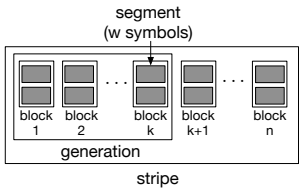

Figure 2: Illustration of notations: hierarchy of stripe, generation, block, and segment, where block 1 to k are data blocks and block $k+1$ to n are parity blocks.

2 Background

Suppose that given k data blocks, we will have r parity blocks under some erasure code. If any k of all the $k+r$ blocks can decode the original data, such erasure codes make the optimal usage of storage space, *e.g.,* RS codes. A block contains a certain number of *symbols*. Typically, a symbol is simply a single byte. The encoding, decoding and reconstruction operations of the erasure code are performed on such symbols with the arithmetic of the so-called *finite field*. In this paper, however, we do not rely on any direct knowledge of the finite field and readers can simply consider its arithmetic as usual arithmetic.

Given the original data, we can divide them into *generations* such that each generation contains k blocks (f_i, $i = 1,\ldots,k$) with the same size. For simplicity, we only consider one generation in this paper, as all generations will be encoded in the same way. Depending on the erasure codes, a block may consist of one or multiple *segments*, where each segment is a row vector of w symbols. For simplicity, we assume that one symbol is a byte in this paper. In other words, each segment contains w bytes. Assuming that each block contains α segments, each block has αw bytes, and we regard f_i as an $\alpha \times w$ matrix. Let $n = k+r$. Given the k original blocks, erasure codes use an $n\alpha \times k\alpha$ *generating matrix* G to compute all blocks in a stripe, *i.e.,* $G \cdot \left[f_1^T \ \cdots \ f_k^T\right]^T$, where f_i^T denotes the transpose of f_i. We can divide G into n submatrices of size $\alpha \times k\alpha$ such that $G = \left[g_1^T \ \cdots \ g_n^T\right]^T$. Therefore, the n blocks generated by G can be represented as $g_i F$ (or block i, for simplicity), $i = 1,\ldots,n$, where $F = \left[f_1^T \ \cdots \ f_k^T\right]^T$. The n blocks computed from the same generation belong to the same stripe. We illustrate in brief the notations described above in Fig. 2.

If the first $k\alpha$ rows of G form an identity matrix,

g_iF is identical to f_i, $i = 1,\ldots,k$. In this way, we can term $g_1F,\ldots,g_kF$ as data blocks and the rest as parity blocks. Erasure codes described in this paper are systematic codes. Typically, there is only one segment in each block under RS codes, *i.e.,* $\alpha = 1$. We can construct RS codes by letting the rest of G be a Cauchy matrix. Given any k blocks in the same stripe, we can have their corresponding submatrix of the generating matrix and then decode the original data by multiplying the inverse of this submatrix with these k blocks.

Under MSR codes, on the other hand, a block contains $d-k+1$ segments (*i.e.,* $\alpha = d-k+1$), where d is the number of available blocks required to reconstruct a missing one, $d > k$. During reconstruction, we call these d blocks as *helpers* and the one being reconstructed as a *newcomer*. To reconstruct any newcomer, we do not need to decode it like under RS codes, but reconstruct it with fractions of d helpers in the same stripe. For example, to reconstruct g_iF, we will compute one segment $v_i^T g_jF$, from block g_jF where v_i is a column vector of size α symbols. With the d segments obtained from helpers, we can reconstruct g_iF by multiplying an $\alpha \times d$ matrix with these d segments.

There have been several constructions of MSR codes (*e.g.,* [11]). In this paper, we will construct our Beehive codes based on one particular *product-matrix* construction proposed by Rashmi *et al.* [11], because 1) the MSR codes constructed are systematic; and 2) unlike other constructions that impose constraints on specific values of d or k, the construction proposed in [11] is much more general by only requiring $d \geq 2k-2$.[2]

We construct Beehive on top of the product-matrix MSR codes, and we exploit an important property of this construction in Beehive. In the construction of MSR codes with given k, r, d, we can explicitly obtain a constant vector $(\lambda_1 \ \ldots \ \lambda_n)$, and an $n \times (k-1)$ matrix A in which every $k-1$ rows are linearly independent. We refer to the i-th row of A as $A_i = (a_{i,1} \ \ldots \ a_{i,k-1})$. Besides, in A the first $k-1$ rows, *i.e.,* $A_i,\ldots,A_{k-1}$, are standard bases. When $i, j \geq k$, to reconstruct a newcomer g_jF, there exists a column vector $\hat{v}_{i,j}$ of size α such that the segment $v_j^T g_iF$ satisfies the following equation:

$$(\lambda_i - \lambda_j)^{-1} v_j^T g_iF = \sum_{l=1}^{k-1} a_{i,l} \cdot (\lambda_l - \lambda_j)^{-1} v_j^T g_lF + \hat{v}_{i,j}^T g_jF, \tag{1}$$

where $\hat{v}_{i,j}$ is a vector of size α symbols that is linearly independent with any other $\alpha - 1$ of such vectors with different values of i and the same value of j.[3]

[2]It is proved in [13] that there exists no construction of such MSR codes if $d < 2k-3$.

[3]This property can be directly derived from Lemma 11 and its proof in [11].

3 Beehive Codes

3.1 Code Construction

Under Beehive, we assume that t newcomers are reconstructed simultaneously, $t > 1$. Unlike traditional erasure codes like RS or MSR codes, we divide one generation of the original data into two parts that contains k and $k-1$ blocks, respectively. In the first part, each block contains $d-k+1$ segments, and $t-1$ segments in the second part. We denote the k blocks in the first part as $F = [f_1^T \ \cdots \ f_k^T]^T$ and the $k-1$ ones in the second part as $C = [c_1^T \ \cdots \ c_{k-1}^T]^T$. We illustrate this process in Fig. 3.

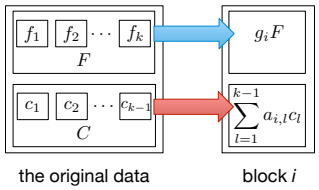

Figure 3: The construction of Beehive codes.

As described in Sec. 2, given k, r, d where $d \geq 2k-2$, we can construct a generating matrix G of the product-matrix MSR code. Then we encode the first part F with such G and get g_iF, $i = 1,\ldots,n$. Along with the product-matrix MSR code, we can also obtain the corresponding matrix A, and then we encode data in the second part as $\sum_{l=1}^{k-1} a_{i,l}c_l$, $i = 1,\ldots,n$.

From the original data, Beehive computes n blocks. Each block contains the $d-k+1$ segments from g_iF and the $t-1$ segments from $\sum_{l=1}^{k-1} a_{i,l}c_l$. Therefore, in the first $k-1$ blocks we can find C directly, since the first $k-1$ rows of A are standard bases. Since the MSR code we use in the first part is systematic, we can also find F directly from the first k blocks. Therefore, Beehive codes are systematic.

On the other hand, there are $d-k+t$ segments in each block under Beehive codes. If each segment contains w symbols, the original data should be grouped into generations of $[k(d-k+t)-(t-1)]w$ symbols. With the same block size, this will lead to a reduction of $(t-1)w$ symbols in each generation under Beehive codes, compared to the theoretical optimum [15] (we omit the proof due to the page limit). Considering the total amount of data in a generation, this loss of storage efficiency is insignificant.

3.2 Decoding and Reconstruction

To decode the original data from any k blocks in a stripe, both F and C must be decoded. Apparently F can be decoded from any k blocks by MSR codes. On the other hand, given k blocks, we have k linear combinations of c_i, $i = 1,\ldots,k-1$. Since any $k-1$ rows in A are linearly independent, c_i can be recovered from any $k-1$ blocks.

Now we consider the reconstruction operation. Without loss of generality, we only show the case of $t = 2$. Let N be the set of newcomers and H be the set of helpers, where $|N| = t$, $|H| = d$, and $N \cap H = \emptyset$.

For any $i \in H$, the helper i computes $(\lambda_i - \lambda_j)^{-1} v_j^T g_i F + u_j^T \sum_{l=1}^{k-1} a_{i,l} c_l$, $i = 1, \ldots, n$, and sends it to newcomer j , where u_j is a vector of size $t-1$ symbols. Any $t-1$ vectors in $\{u_j | j \in \{1, \ldots, n\}\}$ must be linearly independent.

At the side of newcomers, we divide their operation into two stages. Taking newcomer j for example, where $j \geq k$ (we also omit the proof for $j < k$ due to the page limit), in the first stage, it will receive d segments from helpers in H. By (1), each segment can be written as

$$\begin{aligned}&(\lambda_i - \lambda_j)^{-1} v_j^T g_i F + u_j^T \sum_{l=1}^{k-1} a_{i,l} c_l \\ =&\sum_{l=1}^{k-1} a_{i,l}((\lambda_l - \lambda_j)^{-1} v_j^T g_l F + u_j^T c_l) + \hat{v}_{i,j}^T g_j F.\end{aligned}$$

In this way, $a_{i,l}((\lambda_l - \lambda_j)^{-1} v_j^T g_l F + u_j^T c_l)$ is of rank 1, $l = 1, \ldots, k-1$, and $\hat{v}_{i,j}^T g_j F$ is of rank $d - k + 1$, $i \in H$. Thus we can use d segments received from the d helpers to solve $g_j F$, $\forall j \in N$. Meanwhile, $(\lambda_l - \lambda_j)^{-1} v_j^T g_l F + u_j^T c_l$ can be solved as well, $l = 1, \ldots, k-1$.

In the second stage, the newcomer j will send

$$\begin{aligned}&\sum_{l=1}^{k-1} a_{j',l}((\lambda_l - \lambda_j)^{-1} v_j^T g_l F + u_j^T c_l) + \hat{v}_{j',j}^T g_j F \\ =&(\lambda_{j'} - \lambda_j)^{-1} v_j^T g_{j'} F + u_j^T \sum_{l=1}^{k-1} a_{j',l} c_l \quad \text{(from (1))}\end{aligned}$$

to another newcomer j', and it will receive $t-1$ segments from other newcomers as well. Then the newcomer j' can cancel out $(\lambda_{j'} - \lambda_j)^{-1} v_j^T g_{j'} F$ as it has already solved $g_{j'} F$. Then we can get $u_j^T \sum_{l=1}^{k-1} a_{j',l} c_l$, $j \in N \setminus \{j'\}$, and solve $\sum_{l=1}^{k-1} a_{j',l} c_l$ as well, from the $t-1$ segments received from other newcomers. Therefore, it is recommended that the matrix composed by $\{u_j | j \in \{1, \ldots, n\}\}$ contains an identity submatrix, such that this operation can be simplified. In this way, we can reconstruct both $g_{j'} F$ and $\sum_{l=1}^{k-1} a_{j',l} c_l$. During reconstruction, each newcomer will receive $d + t - 1$ segments, achieving the optimal network transfer [15] to reconstruct t blocks.

4 Preliminary Results

We implement Beehive in C++, using the Intel storage acceleration library (ISA-L) [1] for the finite field arithmetic. We also implement RS codes and MSR codes with ISA-L, for comparison purposes.

We let $k = 6$ and $r = 6$ and set the block size to be 60 MB. We first compare the speed of different operations of RS, MSR ($d = 10$), and Beehive codes ($d = 10, t = 2$) in Fig. 4, running on Amazon EC2 instances of type c4.2xlarge. We observe that the encoding and decoding operations of MSR codes and Beehive codes are quite close to each other. The encoding operation of Beehive codes is a bit slower than MSR codes as Beehive codes are built on top of MSR codes. Nevertheless, we do not observe that the speed of decoding or reconstruction operation is significantly different between MSR codes and Beehive codes. The first stage of the reconstruction at the side of newcomers under MSCR codes has lower throughput. However, since only one segment is produced in this stage, the real processing time is much faster than the second stage. This is more important to distributed storage systems as data will be encoded only once but decoded or reconstructed many times. RS codes, on the other hand, can enjoy a higher throughput as there is only one segment in each block under RS codes.

With each block of size 60 MB, a generation under RS or MSR codes is 360 MB, but a generation under Beehive codes is 10 MB less. In fact, with the same storage overhead (*i.e.,* the same k and r), Beehive codes store less actual data. In other words, with the same amount of original data, we may have more generations under Beehive codes. However, this additional storage overhead is marginal. It is just 2.8% in this case.

We compare the disk read and network transfer during reconstruction for RS, MSR, and Beehive codes in Fig. 5. In particular, when $t = 1$, Beehive codes will be the same as MSR codes. When $t > 1$, RS codes and MSR codes reconstruct t blocks separately. In other words, there will be data read during each reconstruction. Hence, as shown in Fig. 5a, Beehive codes achieve much less disk read during reconstruction, which does not change with t, but only depends on the number of helpers. Compared with MSR codes, Beehive codes can save up to 75% of disk read. With regards to network transfer, we observe in Fig. 5b that RS codes always impose the most network transfer. Both MSR codes and Beehive codes require less network transfer with an increasing of d. Beehive

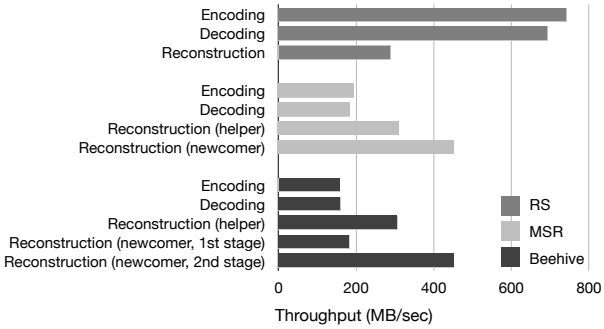

Figure 4: Comparison of the speed of encoding, decoding and reconstruction operations for RS, MSR, and Beehive codes.

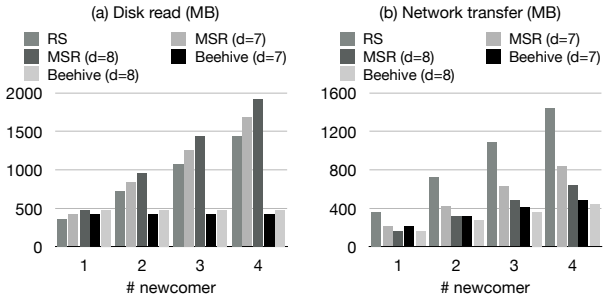

Figure 5: Comparison of disk I/O and network transfer during reconstruction with $k = 6$ and $r = 6$, for RS, MSR, and Beehive codes.

codes, however, can further save network transfer by up to 42.9% against MSR codes and by 69.4% against RS codes. The reduction of network transfer becomes even more significant with an increasing of t.

5 Related Work

In order to optimize network transfer during reconstruction without loss of fault tolerance, Dimakis *et al.* [4] explored the theoretical lower bound of network transfer of the single-block reconstruction, and there are a lot of literatures that present constructions of such erasure codes (called *regenerating codes*) [3]. Among these codes, the *minimum-storage regenerating* (MSR) codes achieve the optimal storage overhead [11]. Network transfer can be further saved when there are multiple blocks to reconstruct at the same time [15]. However, there has been no construction of erasure codes that can achieve this lower bound with general values of parameters and the optimal storage overhead. Shum [15] and Li *et al.* [7] have proposed a construction of such erasure codes that achieve the optimal network transfer with $d = k$ and $t = 2$, respectively. In this paper, we propose a construction that achieves the optimal network transfer with near-optimal storage overhead and a wide range of parameters.

6 Conclusions

In this paper, we propose Beehive, a new family of erasure codes that reconstruct multiple blocks simultaneously and achieve the optimal network transfer with near optimal storage overhead. Through experiments on Amazon EC2, we show that compared to existing erasure codes like RS and MSR codes, Beehive codes can both save network transfer and disk I/O significantly.

Acknowledgements

This paper is supported by the SAVI project and the NSERC Discovery Grant.

References

[1] Intel Storage Acceleration Library. https://01.org/intel%C2%AE-storage-acceleration-library-open-source-version.

[2] BORTHAKUR, D. HDFS Architecture Guide. *Hadoop Apache Project*. http://hadoop.apache.org/common/docs/current/hdfs_design.pdf.

[3] DIMAKIS, A., RAMCHANDRAN, K., WU, Y., AND SUH, C. A Survey on Network Codes for Distributed Storage. *Proceedings of the IEEE 99*, 3 (Mar. 2011), 476–489.

[4] DIMAKIS, A. G., GODFREY, P. B., WU, Y., WAINWRIGHT, M. J., AND RAMCHANDRAN, K. Network Coding for Distributed Storage Systems. *IEEE Trans. Inform. Theory 56*, 9 (2010), 4539–4551.

[5] FORD, D., LABELLE, F., POPOVICI, F. I., STOKELY, M., TRUONG, V.-A., BARROSO, L., GRIMES, C., AND QUINLAN, S. Availability in Globally Distributed File Systems. In *Proc. USENIX Symposium on Operating System Design and Implementation (OSDI)* (2010).

[6] HUANG, C., SIMITCI, H., XU, Y., OGUS, A., CALDER, B., GOPALAN, P., LI, J., AND YEKHANIN, S. Erasure Coding in Windows Azure Storage. In *Proc. USENIX Annual Technical Conference (USENIX ATC)* (2012).

[7] LI, J., AND LI, B. Cooperative Repair with Minimum-Storage Regenerating Codes for Distributed Storage. In *Proc. IEEE INFOCOM* (2014).

[8] MA, A., DOUGLIS, F., LU, G., SAWYER, D., CHANDRA, S., AND HSU, W. RAIDShield: Characterizing, Monitoring, and Proactively Protecting Against Disk Failures. In *Proceedings of the 13th USENIX conference on File and Storage Technologies* (2015).

[9] RASHMI, K. V., NAKKIRAN, P., WANG, J., SHAH, N. B., AND RAMCHANDRAN, K. Having Your Cake and Eating It Too: Jointly Optimal Erasure Codes for I/O, Storage, and Network-bandwidth. In *Proceedings of 13th USENIX Conference on File and Storage Technologies (FAST)* (2015).

[10] RASHMI, K. V., SHAH, N. B., GU, D., KUANG, H., BORTHAKUR, D., AND RAMCHANDRAN, K. A Solution to the Network Challenges of Data Recovery in Erasure-coded Distributed Storage Systems: A Study on the Facebook Warehouse Cluster. In *Proc. 5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)* (2013).

[11] RASHMI, K. V., SHAH, N. B., AND KUMAR, P. V. Optimal Exact-Regenerating Codes for Distributed Storage at the MSR and MBR Points via a Product-Matrix Construction. *IEEE Transactions on Information Theory 57*, 8 (2011), 5227–5239.

[12] SATHIAMOORTHY, M., ASTERIS, M., PAPAILIOPOULOS, D., DIMAKIS, A. G., VADALI, R., CHEN, S., AND BORTHAKUR, D. XORing Elephants: Novel Erasure Codes for Big Data. *Proc. VLDB Endowment* (2013).

[13] SHAH, N., RASHMI, K. V., KUMAR, P., AND RAMCHANDRAN, K. Interference Alignment in Regenerating Codes for Distributed Storage: Necessity and Code Constructions. *IEEE Trans. Inf. Theory 58*, 4 (2012), 2134–2158.

[14] SHAH, N. B., RASHMI, K. V., VIJAY KUMAR, P., AND N, K. Distributed Storage Codes With Repair-by-Transfer and Nonachievability of Interior Points on the Storage-Bandwidth Tradeoff. *IEEE Trans. on Inform. Theory, 58*, 3 (2012), 1837–1852.

[15] SHUM, K. W. Cooperative Regenerating Codes for Distributed Storage Systems. In *Proc. IEEE International Conference on Communications (ICC)* (2011).