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

Jun Li, *Student Member, IEEE,* and Baochun Li, *Fellow, IEEE*

Abstract—In distributed storage systems, erasure codes have been increasingly deployed to tolerate server failures without loss of data. Traditional erasure codes, such as Reed-Solomon codes, suffer from a high volume of network transfer and disk I/O to recover unavailable data at failed storage servers. Typically, unavailable data at different failed storage servers in a distributed storage system are fixed separately. It has been shown that it is possible to reduce the volume of network transfer significantly by reconstructing data from multiple storage servers at the same time. However, there has been no construction of erasure codes to achieve it without imposing strict constraints on system parameters.

In this paper, we propose Beehive codes, designed for optimizing the volume of network transfers to fix the data on multiple failed storage servers. Beehive codes can be constructed over a wide range of system parameters at code rate no more than 0.5, while incurring slightly more storage overhead than Reed-Solomon codes. To achieve the optimal storage overhead as Reed-Solomon codes, we further extend vanilla Beehive codes to MDS Beehive codes, which incurs near-optimal volumes of network transfers during reconstruction. We implement both Beehive and MDS Beehive Codes in C++ and evaluate their performance on Amazon EC2. Our evaluation results have clearly shown that the volume of both network transfers and disk I/O can be conserved by a substantial margin.

Index Terms—distributed storage system, cooperative regenerating codes, MDS, interference alignment.

1 Introduction

Large-scale distributed storage systems, especially those in data centers, store a massive amount of data over a large number of storage servers. As storage servers are built with commodity hardware, their frequent failures can be expected on a daily basis [1]. To keep data available in spite of server failures, replicated data are traditionally stored. For example, the Hadoop Distributed File System (HDFS) [2] stores three copies of the original data (*i.e.*, 3-way replication) on different storage servers by default.

However, there exists an expensive storage overhead to store multiple copies of the original data. With three copies, for example, at most 33% of the total storage space can be effectively used. Therefore, distributed storage systems (*e.g.*, [3], [4]) have been replacing replicated data with erasure codes, especially for cold or archival storage [5]. Using erasure codes, distributed storage systems can enjoy better tolerance against server failures, with the same or even less storage overhead.

Among erasure codes, Reed-Solomon (RS) codes are the most popular choice since RS codes can achieve the optimal storage overhead while tolerating the same number of failures. To achieve failure tolerance with RS codes, we assume that data are stored in *blocks* with a fixed size, a common practice in most distributed storage systems. RS codes can compute r parity blocks from k original data blocks, such that any k of the total $k+r$ blocks can recover all the original data blocks. We can group such $k+r$ blocks as one *stripe*, and blocks in the same stripe are stored in different storage servers. Therefore, RS codes can tolerate at most r failures within the same stripe. For example, RS codes with $k=4$ and $r=2$ can tolerate the same number of failures as 3-way replication, by storing only 6 blocks in total while 3-way replication needs to store 12 blocks.

Once a storage server fails, this failure can be fixed by *reconstructing* the missing data on a replacement storage server. With RS codes, the replacement server needs to download k other existing blocks in the same stripe to reconstruct one single block, imposing k times of the network transfer with replication. It has been reported that in one of Facebook's clusters, the reconstruction operation can incur more than 100 TB of data every day [6]. Besides, the same amount of disk I/O will also be imposed on the existing storage servers as well.

To reduce the amount of network transfer during reconstruction, there have been considerable interests in the construction of *minimum-storage regenerating* (MSR) codes (*e.g.*, [7]) that achieve the optimal network transfer to reconstruct a missing block while consuming the same

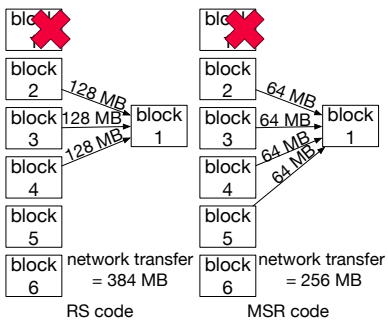


Fig. 1. The amount of network transfer imposed by reconstructing the missing block using RS and MSR codes, with $k=3$ and $r=3$, and a block size of 128 MB.

• *J. Li and B. Li are with the Department of Electrical and Computer Engineering, University of Toronto.*

storage overhead as RS codes. Compared with RS codes, MSR codes can significantly save network transfer during reconstruction. For example, in Fig. 1, we assume that each block has 128 MB and we compute 3 parity blocks with RS codes and MSR codes where $k = 3$ and $r = 3$. To reconstruct one block with MSR codes, the replacement server needs to download only a fraction of each block from four existing storage servers, rather than three whole blocks with RS codes. In this example, though MSR codes require data to be downloaded from one more server than RS codes, the total network transfer can still be saved by 33.3% as from each server only a half of its data is downloaded. However, MSR codes incur even more disk I/O on storage servers, since with MSR codes the data sent to the replacement server typically have to be encoded from the whole block, and the replacement server needs to download data from even more servers than RS codes.

MSR codes are designed to optimize network transfer to reconstruct one single block with the optimal storage overhead. However, in a distributed storage system it is not uncommon that multiple blocks need to be reconstructed with various reasons [8]. First, blocks failures can be correlated in the distributed storage system. It has been reported that disk failures are correlated such that disks installed together are more likely to fail together [9]. When a network failure occurs, on the other hand, multiple servers can be disconnected from the network. Data inside such servers hence become unavailable and need reconstruction. Second, large-scale distributed storage systems scan block failures periodically [5]. Because of the large volume of data stored in the system, the scan can take a long period of time, and it becomes more likely to observe multiple failures. The correlated failures make it even more likely to find multiple failures within the same period of the scan. Third, distributed storage systems may batch reconstructions on purpose, in order to save power [5] or to avoid unnecessarily reconstructing blocks that are just temporarily unavailable [10].

When fixing multiple failures is not uncommon in the distributed storage system or even is intended, we can design erasure codes that reconstruct data from multiple failures in batches rather than separately, such that even less network transfer will be incurred than MSR codes [11], [12]. Meanwhile, we can significantly save disk I/O as we can read existing blocks only once instead of multiple times. However, while the optimal network transfer to reconstruct multiple blocks has been theoretically established [12], there has been no explicit construction of erasure codes that achieve both the optimal network transfer to reconstruct exact data of multiple failed servers and the optimal storage overhead simultaneously, except for those that impose strict constraints on system parameters (*e.g.*, [12], [13]).

In this paper, we propose Beehive codes, a new family of erasure codes, that can reconstruct data of multiple blocks and support a wide range of system parameters with code rate no more than $\frac{1}{2}$. Besides an instant reduction of disk I/O because of reconstructing multiple blocks at the same time, Beehive codes achieve the optimal network transfer of reconstructing multiple blocks. As illustrated in Fig. 2, compared to MSR codes, the total amount of disk read is saved by 50% when we reconstruct two blocks at the same time, and the amount of network transfer is saved by 16.6% as well. In fact, the more failures we fix at the same time, the more network transfer can be saved.

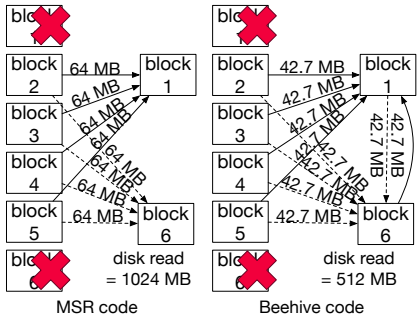


Fig. 2. Comparison of the amount of network transfer and disk read imposed by the reconstruction of two missing blocks with MSR and Beehive codes, with $k = 3$, $r = 3$, and blocks of size 128 MB.

Nevertheless, Beehive codes are not optimal in terms of the storage overhead, though we show that the additional storage overhead is marginal. In this paper, we also present an extension of Beehive codes, called MDS[1] Beehive codes, that achieve the optimal storage overhead like RS and MSR codes. On the other hand, MDS Beehive codes consume more network transfer than Beehive codes. Yet, we show that the network transfer of MDS Beehive codes is very close to Beehive codes.

We implement both Beehive and MDS Beehive codes in C++ and evaluate their performance on Amazon EC2. Our experimental results show that compared to MSR codes, Beehive codes can save up to 23.8% of network traffic and up to 83.3% of disk I/O during reconstruction. Similar results can be observed with MDS Beehive codes as well.

2 Background

The construction of Beehive and MDS Beehive codes depends on Product-Matrix-MSR (PM-MSR) codes [7] by exploiting one of its important property. In this section, we briefly review the construction of PM-MSR codes and this important property, and survey the related literature as well.

2.1 Model and Notations

Suppose that the original data contain B bytes, which can be grouped into k blocks with the same size (*i.e.*, $\frac{B}{k}$ bytes). Given k, r, and d, the corresponding (k, r, d) MSR codes can encode the k original blocks into $k + r$ blocks. If the MSR codes are *systematic*, k of the total $k + r$ blocks contain the original data, and thus they are known as *data blocks*. The other r blocks, on the other hand, are known as *parity blocks*. Systematic erasure codes are desirable for distributed storage systems, as the original data can be directly read without any decoding operations. We assume that all erasure codes discussed and constructed in this paper are systematic unless mentioned otherwise.

MSR codes achieve the optimal storage overhead because each block contains $\frac{B}{k}$ bytes. In other words, any k blocks have exactly B bytes with MSR codes (as well as all

1. In coding theory, erasure codes that achieve the optimal storage overhead are termed as maximum distance separable (MDS) codes.

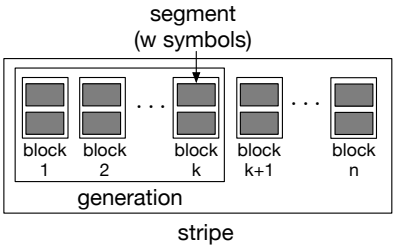


Fig. 3. Illustration of notations: a hierarchy involving stripe, generation, block, and segment, where block 1 to k are data blocks and block $k+1$ to n are parity blocks.

other MDS codes such as RS codes), and it is impossible to recover the original data with less data. The k original blocks and the r parity blocks belong to the same *stripe*, and all blocks in a stripe will be stored in different servers. To reconstruct a block, the replacement server will contact d existing servers that store other blocks in the same stripe. To achieve the optimal network transfer, MSR codes require the replacement server to download $\frac{B}{k(d-k+1)}$ bytes from each of the d servers [14].

To achieve the optimal network transfer, typically a block will have to contain $\gamma(d-k+1)$ *segments* with MSR codes, such that a server only needs to send γ segments to the replacement server during reconstruction, though typically this segment will be computed from all $\gamma(d-k+1)$ segments in the block. On the other hand, since each block should have the same size, we will need to divide the original data into *generations* if there are more than B bytes. For simplicity, we consider one generation in this paper, as all generations will be encoded in the same way. Hence, all blocks we discuss in this paper will be within the same stripe. In Fig. 3, we illustrate the notations described above.

2.2 Product-Matrix-MSR codes

The PM-MSR codes are constructed over a finite field, where data are treated as symbols on the finite field. In practice a symbol on a finite field can simply be a byte.[2] Thus, the operations of encoding, decoding and reconstruction are all performed on bytes with the arithmetic of the 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.

We now briefly introduce the PM-MSR codes to the extent required to understand this paper, and more details can be found in [7]. In the PM-MSR code, $\gamma = 1$. Thus, a block contains $\alpha = d-k+1$ segments, and each segment contains w bytes of data where $w = \frac{B}{\alpha k}$.

We show the construction of PM-MSR codes via an example where $k=3, r=3, d=4$, and thus $\alpha = 4-3+1 = 2$. As $k=3$, we have three original blocks, f_1, f_2, and f_3. Each of these blocks are further divided into 2 segments as $\alpha = 2$, *i.e.*, f_{i1} and f_{i2}, $i = 1,2,3$. For now we assume that each segment contains one symbol only. The corresponding PM-MSR code can then be constructed[3] by multiplying two matrices together, *i.e.*,

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 5 & 15 \\ 1 & 4 & 3 & 12 \\ 1 & 5 & 2 & 10 \\ 1 & 6 & 7 & 1 \end{bmatrix} \cdot \begin{bmatrix} f_{11} & f_{12} \\ f_{12} & f_{21} \\ f_{22} & f_{31} \\ f_{31} & f_{32} \end{bmatrix}, \tag{1}$$

where any 4 rows in the first matrix are linearly independent and the second matrix can be vertically split into two symmetrical submatrices, each of which contains a half of the original data. The result of (1) is a $(k+r)\times\alpha$ matrix where each row corresponds to the $\alpha = 2$ segments in each of the $k+r$ encoded blocks. For example, the first block contains two segments, $f_{11}+f_{12}+f_{22}+f_{31}$ and $f_{12}+f_{21}+f_{31}+f_{32}$.

Traditionally, a erasure code is represented in the form of the generating matrix, which looks different from the form in (1). In fact, we can equivalently write the code constructed in (1) into the form of the generating matrix, *i.e.*,

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 2 & 0 & 4 & 0 & 8 \\ 0 & 1 & 0 & 2 & 4 & 8 \\ 1 & 3 & 0 & 5 & 0 & 15 \\ 0 & 1 & 0 & 3 & 5 & 15 \\ 1 & 4 & 0 & 3 & 0 & 12 \\ 0 & 1 & 0 & 4 & 3 & 12 \\ 1 & 5 & 0 & 2 & 0 & 10 \\ 0 & 1 & 0 & 5 & 2 & 10 \\ 1 & 6 & 0 & 7 & 0 & 1 \\ 0 & 1 & 0 & 6 & 7 & 1 \end{bmatrix} \cdot \begin{bmatrix} f_{11} \\ f_{12} \\ f_{21} \\ f_{22} \\ f_{31} \\ f_{32} \end{bmatrix}. \tag{2}$$

Comparing the result of (2) with (1), the only difference is that we move the two segments of the same block into two consecutive rows in (2). As the code given in the original form of PM-MSR codes in [7] can be equivalently converted into the form with the generating matrix, we always use the generating matrix to describe a PM-MSR code in the rest of the paper.

One benefit of using the generating matrix to describe erasure code is that we can directly extend the size of each segment without changing the generating matrix. In general, we use a horizontal vector f_{ij} of size w bytes to represent the j-th segment in block i, $j = 1,\ldots,\alpha$, and use $f_i,\ldots,f_k$, k matrices of size $\alpha\times w$, to represent the k data blocks. Let $n = k+r$, and then a PM-MSR code can be associated with an $n\alpha\times k\alpha$ *generating matrix* G to generate all blocks in a stripe. If we divide G into n submatrices of size $\alpha\times k\alpha$ such that $G = \begin{bmatrix} g_1^T & \cdots & g_n^T \end{bmatrix}^T$, where g_i^T represents the transpose of g_i, the n blocks generated by G can thus be denoted as g_iF where $F = \begin{bmatrix} f_1^T & \cdots & f_k^T \end{bmatrix}^T$. In the example of (2) we can have $g_1 = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$, and we can obtain g_i from all blocks in this way, $i = 1,\ldots,n$. Given any k blocks, a square submatrix of G can be composed of the $k\alpha$ rows corresponding to these k blocks. Hence, we can

2. As a byte can have at most 2^8 possible values, it corresponds to a finite field of size 2^8. Though the minimum size of the finite field required by MSR codes depends on the values of system parameters, 2^8 is big enough for typical values of system parameters in practice. We can also easily extend the construction of MSR codes to a larger finite field.

3. For simplicity in this example the code is constructed on a small finite field of size 2^4.

decode the k data blocks (*i.e.*, F) by multiplying the inverse of this submatrix with these k blocks.

The above example of PM-MSR codes is not systematic. However, we can easily convert a non-systematic PM-MSR code into a systematic one. Given a generating matrix G, its submatrix G_0 containing the top $k\alpha$ rows in G must be non-singular as the corresponding k blocks it generates can decode the original data. Hence, we can always get $G_0^{-1}F$ from F and *vice versa*, and we can use G to encode $G_0^{-1}F$, which is equivalent to encoding F with a new generating matrix GG_0^{-1}, *i.e.*, $G \cdot (G_0^{-1}F) = (GG_0^{-1}) \cdot F$. No change will occur to the performance such as failure tolerance or network traffic during reconstruction, as all corresponding operations remain the same. Using this technique termed as *symbol remapping* in [7], [15], we can obtain a systematic PM-MSR code as the top k rows in GG_0^{-1} is an identity matrix. In other words, g_iF is identical to the data block f_i, $i = 1, \ldots, k$. Thus, the first k blocks are data blocks and the rest are party blocks. In the rest of this paper, the PM-MSR codes we refer to will always be systematic where the top $k\alpha$ rows in G form an identity matrix.

Similarly, we use the generating matrix to describe RS codes as well. Given k and r, the generating matrix G of the corresponding RS code will be a $(k + r) \times k$ matrix. With RS codes, each block contains 1 segment only, and hence each block can similarly be denoted as g_iF. The generating matrix G can be constructed by a matrix in which any k rows are linearly independent, *e.g.* a Cauchy matrix [16].

2.3 Interference alignment

To reconstruct a block g_iF, we need to have d existing blocks, which we term as *helpers* during reconstruction, and compute one segment on each helper. With PM-MSR codes, it is required that $d \geq 2k - 2$. We show an example of the reconstruction in Fig. 4, where the PM-MSR code is the systematic version of the code in (2) with $k = 3, r = 3, d = 4$. Assume that block 1 is unavailable, and we choose $d = 4$ blocks from the other 5 blocks as helpers. We show that each chosen block simply adds its two segments into one and block 1 can be reconstructed from these 4 segments.

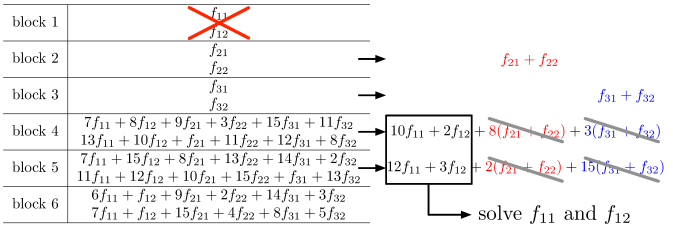


Fig. 4. An illustration of reconstructing one missing block encoded by PM-MSR codes with $k = 3, r = 3, d = 4$. With 4 segments computed from $d = 4$ existing blocks, we cancel the interference components $f_{21} + f_{22}$ and $f_{31} + f_{32}$ in the last two segments and solve the desired components f_{11} and f_{12}.

We notice that in the 4 segments offered by the existing blocks, there are some components, termed as *interference components*, that do not contribute to block 1 including $f_{21}, f_{22}, f_{31}, f_{32}$, since block 1 only requires two segments, f_{11} and f_{12}, which are called *desired components*. Hence, in order to reconstruct block 1, we need to get rid of the interference components from the 4 received segments.

Though we can solve both the 4 interference components and the 2 desired components by 6 linearly independent segments, PM-MSR codes achieve the optimal network traffic during reconstruction by requiring only 4 segments, because the interference components are carefully *aligned*. We can see from Fig. 4 that the linear combinations of f_{21} and f_{22} in the 4 segments are either 0 (non-existing) or a scalar multiple of $f_{21} + f_{22}$, and we can observe the same pattern for f_{31} and f_{32}. As we do not need the interference components to reconstruct the missing block, we do not have to solve them at all. Instead, we can simply solve $f_{21} + f_{22}$ and $f_{31} + f_{32}$ in this case.

This property is known as *interference alignment*, a concept borrowed from the context of wireless communication [17] that aligns the interference components to a low-dimensional linear subspace to extract desired components. Similarly, in the reconstruction of PM-MSR codes, interference alignment makes it possible for PM-MSR codes to reconstruct a block exactly from the segments received from helpers with the optimal network transfer, and plays an important role in the construction of our own codes as well. We now formalize this property to facilitate our future code construction.

We use D to represent the index of each helper. For example, $D = \{1, 2\}$ if we have two helpers g_1F and g_2F. With PM-MSR codes, each helper will compute this segment with a vector v_i to reconstruct g_iF. In other words, we will compute $v_i^T g_j F$ from a helper g_jF. For example, we have $v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ in Fig. 4. Then we can reconstruct g_iF from d segments $v_i^T g_j F$, $j \in D$.

For convenience, we start from the case that $d = 2k - 2$. At this time, $\alpha = k - 1$. With PM-MSR codes, any α columns in $V = [v_1 \ \ldots \ v_n]$ are linearly independent, where typically V is constructed as a Vandermonde or Cauchy matrix. Let L be an α-subset of $\{1, \ldots, n\}$, and then any v_i can be represented as

$$v_i = \sum_{l \in L} a_{i,l} v_l. \tag{3}$$

On the other hand, from the construction of PM-MSR codes, we can explicitly obtain a vector $\Lambda = [\lambda_1, \ldots, \lambda_n]$. If g_jF is the block to be reconstructed where $j \notin L$, we can let $u_{i \to j} = (\lambda_i - \lambda_j)^{-1} v_j$, and it is proved in [7] that

$$u_{i \to j}^T g_i F = \sum_{l \in L} a_{i,l} u_{l \to j}^T g_l F + \tilde{u}_{i \to j}^T g_j F, \tag{4}$$

where $\tilde{u}_{i \to j} = (\lambda_i - \lambda_j)^{-1} v_i - \sum_{l=1}^{k-1} a_{i,l} (\lambda_l - \lambda_j)^{-1} v_l$, $i \neq j$.

From (4), we can see that the components with g_lF are interference components if $l \in L$, and the component with g_jF is the desired component. It is easy to see that $\forall i \in D$, interference components in (4) with the same l are aligned such that they are scalar multiples of each other. Therefore, we can solve g_jF with segments received from $\alpha + (k-1) = d$ helpers. Besides, notice that with a given i and j not in L, the coefficient, $a_{i,l}$, does not change for all $l \in L$.

For the more general case with $d > 2k - 2$, (k, r, d) PM-MSR codes are constructed from $(k + x, r, d + x)$ PM-MSR codes where $x = d - (2k - 2)$. The technique to convert the code is to assume the first x blocks of original data are zeros, and then all elements corresponding to the x blocks in the

generating matrix as the eventual result will not be affected. In other words, suppose that $\hat{G}$ is the generating matrix of the $(k+x, r, d+x)$ PM-MSR codes. By the definition above, we know that $\hat{G}$ has $(n+x)\alpha$ rows and $(k+x)\alpha$ columns. The (k, r, d) PM-MSR codes will be constructed by truncating the first $x\alpha$ rows in $\hat{G}$, and its generating matrix G is a submatrix of $\hat{G}$ with the last $n\alpha$ rows and the last $k\alpha$ columns.

To reconstruct a block, the interference alignment defined in (4) still applies. However, to find the coefficients $a_{i,l}$ the truncation in the code construction has to be considered. With the $(k+x, r, d+x)$ PM-MSR codes, we can obtain the matrix $\hat{V} = [\hat{v}_1 \ \ldots \ \hat{v}_{n+x}]$, and each $\hat{v}_i$ can be expressed as $\sum_{i \in \hat{L}} \hat{a}_{i,l}\hat{v}_l$, where $\hat{L}$ is an α-subset of $\{1, \ldots, n+x\}$. We can obtain $\hat{u}_{i \to j}$ from the $(k+x, r, d+x)$ PM-MSR codes as well, $i \neq j$.

In the reconstruction with the (k, r, d) PM-MSR codes, suppose that L is the set of helpers, and then we define $L_x = \{l + x | l \in L\}$, and $\hat{L} = \{1, \ldots, x\} + L_x$. A helper $g_i F$ is going to send $u_j^T g_i F$ to the replacement server, where $u_{i \to j} = \hat{u}_{(i+x) \to (j+x)}$. Therefore, we can write the same formula as (4), where $a_{i,l} = \hat{a}_{i+x,l+x}$, and $\tilde{u}_{i \to j} = (\hat{\lambda}_{i+x} - \hat{\lambda}_{j+x})^{-1}\hat{v}_{i+x} - \sum_{l \in \hat{L}} \hat{a}_{i+x,l}(\hat{\lambda}_l - \hat{\lambda}_{j+x})^{-1}\hat{v}_l$.

This property of interference alignment of PM-MSR codes will serve as a foundation in the construction of Beehive and MDS Beehive codes in this paper.

2.4 Related Work

In order to optimize network transfer during reconstruction without loss of fault tolerance, Dimakis *et al.* [14] have explored the theoretical lower bound of network transfer of single-block reconstruction, and there are many related works that present constructions of such erasure codes (called *regenerating codes*) [18]. Among these codes, *minimum-storage regenerating* (MSR) codes achieve the optimal storage overhead as RS codes.

Besides the two system parameters, k and r, of RS codes, MSR codes introduce one more system parameter d, $d \geq k$. When $d = k$, MSR codes simply need to obtain k whole blocks to reconstruct one unavailable block, just the same as RS codes, and hence in this paper we always assume $d > k$ unless mentioned otherwise. Typically, with MSR codes it will only need to receive fractions of d blocks, instead of whole blocks, to reconstruct an unavailable block. In fact, the higher value d is, the less network transfer will be consumed during reconstruction.

There has been various literature that proposes constructions of MSR codes. While some constructions of MSR codes do not reconstruct blocks exactly but only preserve the ability to tolerant failures (*e.g.*, [14], [19]), in this paper we only consider the constructions (*e.g.*, [7]) that the unavailable blocks can be reconstructed exactly, otherwise data blocks will not contain original data after reconstruction. As the distributed storage system favors accessing original data without decoding, it is required to reconstruct the unavailable block exactly. Besides, among MSR codes that can reconstruct data exactly, there exist two categories that reconstruct all blocks exactly and reconstruct only data blocks exactly (*e.g.*, [20], [21]). In this paper, our attentions are paid at those that reconstruct all blocks exactly.

Typically, an MSR code encodes k blocks into $k+r$ blocks where each block contains α *segments*. During reconstruction, a replacement server obtains γ segments from each of d available blocks, where $\alpha = \gamma(d-k+1)$. A code with a smaller value of γ will make encoding, decoding, and reconstruction of less complexity. The complexity is hence minimized when $\gamma = 1$. However, Shah *et al.* [22] have proved that there exists no construction of MSR codes with $\gamma = 1$ when $d < 2k-3$. When $\gamma = 1$, a construction of MSR codes with $d = k+r-1 \geq 2k-1$ was proposed by Suh *et al.* [23], and Rashmi *et al.* [7] have pushed one step further by proposing the Product-Matrix-MSR codes with $d \geq 2k-2$, which is the most general construction in terms of the values of system parameters for MSR codes with $\gamma = 1$ to our best knowledge. With $d \geq 2k-2$ and $d < k+r$, the maximum code rate of such MSR codes is $\frac{k}{2k-1}$.

When a higher complexity is permitted, MSR codes with a more general range of system parameters can be constructed that achieve code rates high than $\frac{1}{2}$. Cadambe *et al.* [24] have shown that MSR codes can be asymptotically constructed for all valid (k, r, d) when γ goes to infinity. Reconstructing only data blocks exactly, Goparaju *et al.* [21] have proposed a construction of MSR codes for all valid values of (k, r, d) with γ exponential in k. As for MSR codes that reconstruct any block exactly with a finite value of γ, constructions were proposed in [25], [26], [27], [28], [29] where γ is still exponential in k. Polynomial γ exists in some particular construction of MSR codes with some fixed code rate [30], [31], [32].

On the other hand, to reconstruct a block with MSR codes, typically the γ segments have to be encoded from all α segments in the block, and thus MSR codes will incur more disk I/O than RS codes as more blocks will be contacted with MSR codes (*i.e.*, $d > k$). As the optimal disk I/O is achieved when the data read is the same as data transferred over the network, some constructions of MSR codes with $r = 2$, such as Zigzag codes [26] and some other variants [27], [28], [29], can reconstruct data blocks exactly with the optimal disk I/O. Rashmi *et al.* [15] have also proposed a variant of the Product-Matrix-MSR codes that support to reconstruct $d-k+1$ blocks with the optimal disk I/O. However, in general MSR codes do not optimize disk I/O for reconstructing any unavailable block.

Network transfer and disk I/O can be further saved when there are multiple blocks to reconstruct at the same time. It is shown that *cooperative regenerating codes* [11], [12], [13], [33], [34], [35], [36], [37] can reconstruct multiple blocks with even lower network transfer than regenerating codes. Similarly, *minimum-storage* cooperative regenerating (MSCR) codes achieve the optimal storage overhead.

Besides k, r, and d, MSCR codes introduce one more system parameter t such that t blocks are reconstructed at the same time. Similarly, MSCR codes can be constructed asymptotically as γ goes to infinity [24] where $\alpha = (d-k+t)\gamma$. However, there has been no explicit construction of MSCR codes that can achieve the optimal network transfer during reconstruction, with general values of system parameters and finite values of γ. So far there exist only a few constructions of MSCR codes with specific values of some system parameters, such as $d = k$ [12], $k = 2$ [33], $t = 2$ [13], [35], [36], and $d = n - t$ [37]. In this paper,

we propose Beehive codes that achieve the same network transfer of MSCR codes, with near-optimal storage overhead over a wide range of system parameters. The construction of Beehive codes aims at low code rates (at most $\frac{1}{2}$), a low complexity with $\gamma = 1$, and reconstructing any t blocks exactly. MDS Beehive codes, an extension of Beehive codes, achieve the optimal storage overhead with near-optimal network transfer. Though neither Beehive codes nor MDS Beehive can be strictly categorized as MSCR codes, due to the non-optimality of either storage overhead or network transfer, both Beehive codes and MDS Beehive codes offer a more practical solution for distributed storage systems than all existing constructions of MSCR codes, as they can be constructed with much more combinations of system parameters with performance very close to MSCR codes.

3 Beehive Codes

3.1 Code Construction

Beehive codes are associated with four system parameters: k, r, d, t. With given system parameters, (k, r, d, t) Beehive codes encode data into $n(= k + r)$ blocks where any k of them can recover the original data. The reconstruction operation requires d helpers to fix data of t unavailable blocks, $t > 1$.

Unlike other erasure codes, such as RS codes or MSR codes, with Beehive codes a generation of data will firstly be grouped before encoding, into two parts that contain k and $k-1$ blocks, respectively. Blocks in the two parts, however, will have different sizes. Each block contains $d - k + 1$ segments in the first part, and $t - 1$ segments in the second part, where segments in the two parts have the same size. As shown in Fig. 5, we denote the k blocks in the first part as $F = [f_1^T \ \cdots \ f_k^T]^T$ and the $k-1$ blocks in the second part as $C = [c_1^T \ \cdots \ c_{k-1}^T]^T$.

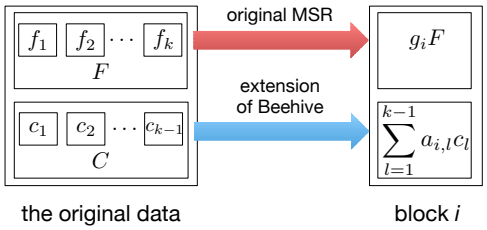


Fig. 5. The construction of Beehive codes. To compute a block, Beehive codes combine two parts of data encoded with MSR codes and RS codes.

We construct Beehive codes by combining MSR and RS codes on these two parts, where F will be encoded by PM-MSR codes and C will be encoded by RS codes. As described in Sec. 2.2, we can construct the generating matrix G of (k, r, d) PM-MSR codes when $d \geq 2(k-1)$. Hence, we can encode F with such $G = [g_1^T \ \cdot \ g_n^T]^T$ and get g_iF, $i = 1, \ldots, n$, and thus Beehive codes require that d must be no less than $2k-2$ as well.

As for the second part C, we encode the $k-1$ blocks with RS codes where coefficients are obtained from the (k, r, d) PM-MSR codes used in the first part. As described in Sec. 2.3, we can let $L = \{1, \ldots, k-1\}$, and get $a_{i,l}$, $i = 1, \ldots, n$, $l = 1, \ldots, k-1$. Hence we encode the second part into n blocks as $\sum_{l=1}^{k-1} a_{i,l}c_l$, $i = 1, \ldots, n$.

Given data encoded from the two parts, Beehive codes finally group them into n blocks such that each block contains $d-k+1$ segments from g_iF and $t-1$ segments from $\sum_{l=1}^{k-1} a_{i,l}c_l$, as shown in Fig. 5. Therefore, with Beehive codes, each block will contain $d-k+t$ segments. If each segment contains w bytes, each block is of size $(d-k+t)w$ bytes, and each generation will have $[k(d-k+1)+(k-1)(t-1)]w$ bytes.

It is easy to see that any k blocks among n blocks computed with Beehive codes can recover the original data. To decode the original data, we need to decode both F and C. Given any k blocks, we can decode F from g_iF with MSR codes. On the other hand, let A be the matrix composed of $a_{i,l}$, *i.e.*, $A = [a_{i,l}]_{n\times(k-1)}$. By (3) it can be inferred that any $k-1$ rows in A are linearly independent since any α vectors in V are linearly independent. Hence, we can decode C from any $k-1$ blocks. Therefore, we can decode both F and C from any k blocks.

With Beehive codes, the original data are embedded into the first k blocks. Since PM-MSR codes are systematic codes, g_iF in the first k blocks ($1 \leq i \leq k$) contains the original data in F. On the other hand, since $a_{i,l}$ is obtained as coefficients in (3), we can find the original data of C in the first $k-1$ blocks with Beehive codes. Therefore, Beehive codes are systematic codes.

However, Beehive codes do not achieve the optimal storage overhead like RS or MSR codes. The amount of data in any k blocks is $k(d-k+t)w$ bytes, which are $(t-1)w$ bytes more than the original data in each generation. On the other hand, we can show that this additional storage overhead is marginal. Given k and r, the storage overhead of RS or MSR codes is $\frac{n}{k}$, while the storage overhead of Beehive codes is

$$\frac{n(d-k+t)}{k(d-k+t)-(t-1)} \in \left(\frac{n}{k}, \frac{n}{k-1}\right). \tag{5}$$

With practical values of k, the upper bound in (5) is close to the optimum. For example, when $k = 10$, Beehive codes will increase the storage overhead by at most 11.1%. In other words, with the same amount of original data, Beehive codes will incur 11.1% more storage space to tolerate the same number of failures.

Moreover, Beehive codes require that $d \geq 2k-2$ as the construction is built upon PM-MSR codes. Hence, the code rate of Beehive codes, which is the inverse of the storage overhead mentioned above, will be less than $\frac{k}{n} \leq \frac{k}{d+t} \leq \frac{1}{2}$ when $t > 1$.

3.2 Reconstruction

Now we discuss how to reconstruct multiple blocks with Beehive codes. We assume that N is the set of indexes of blocks to be reconstructed, and D is the set of indexes of helpers, where $|N| = t$, $|D| = d$, and $N \cap D = \varnothing$. For convenience, we term blocks to be reconstructed as *newcomers*, and we may use the index of a helper/newcomer to represent the helper/newcomer itself in this paper.

With Beehive codes, a block contains g_iF and $\sum_{l=1}^{k-1} a_{i,l}c_l$. For any $i \in D$ and $j \in N$, the helper i will send one segment $u_{i\to j}^T g_iF + h_j^T \sum_{l=1}^{k-1} a_{i,l}c_l$ to the newcomer j, where h_j is a vector of size $t-1$ bytes. The only requirement

of h_j is that any $t-1$ vectors in $\{h_j|1 \le j \le n\}$ must be linearly independent.

At the side of the newcomer, its operation is divided into two stages. In the first stage, each newcomer will send one segment to each of other newcomers. In this stage, we will exploit the interference alignment of the PM-MSR codes in (4). Since all coefficients $a_{i,l}$ used in the construction of Beehive codes is obtained when $L = \{1, \ldots, k-1\}$, we first assume that for all $j \in N$, $j \notin L$, and then extend into the general case of any newcomers.

Given a newcomer j during reconstruction, it receives d segments from all helpers, *i.e.*, $u_{i\to j}^T g_i F + h_j^T \sum_{l=1}^{k-1} a_{i,l} c_l$, $i \in D$. By (4), each such segment can be written as

$$u_{i\to j}^T g_i F + h_j^T \sum_{l=1}^{k-1} a_{i,l} c_l = \sum_{l=1}^{k-1} a_{i,l}(u_{l\to j}^T g_l F + h_j^T c_l) + \tilde{u}_{i\to j} g_j F. \quad (6)$$

Thus, similar to PM-MSR codes, we can consider $u_{l\to j}^T g_l F + h_j^T c_l$, $1 \le l \le k-1$, as interference components, and each interference component is aligned with different values of i. As $|D| = d$, we can solve $g_j F$ as well as the $k-1$ interference components. The interference alignment will be used in the second stage, to make the newcomer reconstruct $\sum_{l=1}^{k-1} a_{j,l} c_l$.

Taking newcomer j' as an example, the newcomer j will send the newcomer j' a segment linearly combined from $g_j F$ and interference components above, *i.e.*, $\sum_{l=1}^{k-1} a_{j',l}(u_{l\to j}^T g_l F + h_j^T c_l) + \tilde{u}_{j'\to j}^T g_j F$. Again, by (4), we can get

$$\sum_{l=1}^{k-1} a_{j',l}(u_{l\to j}^T g_l F + h_j^T c_l) + \tilde{u}_{j'\to j}^T g_j F = u_{j'\to j}^T g_{j'} F + h_j^T \sum_{l=1}^{k-1} a_{j',l} c_l. \quad (7)$$

As the newcomer j' has solved $g_{j'} F$ in the first stage, $u_{j'\to j}^T g_{j'} F$ can be canceled out. From all the other $t-1$ newcomers, the newcomer j' can get $t-1$ segments $h_j^T \sum_{l=1}^{k-1} a_{j',l} c_l$, $\forall j \in N \setminus \{j'\}$. Since any $t-1$ vectors in $\{h_j|j \in N\}$ are linearly independent, in the second stage, any newcomer j' can thus solve $\sum_{l=1}^{k-1} a_{j',l} c_l$, $j' \in N$. Therefore, the two parts in a Beehive block can be reconstructed from these two stages.

Now we discuss a more general case when there can be newcomer j between 1 and $k-1$. To deal with this case, we will equivalently transform Beehive codes with a different L such that $N \cap L = \varnothing$. Since $d > k-1$, we can let $L = (l_1 \ \cdots \ l_{k-1})$ be a $(k-1)$-subset of D. Since $N \cap D = \varnothing$, any newcomer in D will not be an element of L as well. We will prove that after such a transformation, similar interference alignment can be applied.

Since in the first $k-1$ blocks we can find the original data of c_i, $1 \le i \le k-1$, we can let $c_i = \sum_{l=1}^{k-1} a_{i,l} c_l$, $1 \le i \le n$. We also let $A_i = [a_{i,1} \ \cdots \ a_{i,k-1}]^T$, $1 \le i \le n$. Given L, we define A_L as the matrix containing A_i for all i in L, *i.e.*, $A_L = [A_{l_1} \cdots A_{l_{k-1}}]^T$. Then we can write c_i as

$$c_i = \sum_{l=1}^{k-1} a_{i,l} c_l = \sum_{m=1}^{k-1} a'_{i,m} c_{l_m}, \quad (8)$$

where we let $A'_i = [a'_{i,1} \ \cdots \ a'_{i,k-1}]^T = \left(A_i^T \cdot A_L^{-1}\right)^T$.

On the other hand, consider the interference alignment of PM-MSR codes in Sec. 2.3. We know that $\hat{v}_{l_i+x} = \sum_{l=1}^{k-1+x} \hat{a}_{l_i+x,l} \hat{v}_l$, where $x = d-2(k-1)$. Hence, by letting $\hat{A}_i = [\hat{a}_{i,1} \ \cdots \ \hat{a}_{i,k-1+x}]^T$, we can get

$$\hat{v}_{i+x}^T = \hat{A}_{i+x}^T \cdot \begin{bmatrix} \hat{v}_1^T \\ \vdots \\ \hat{v}_{k-1+x}^T \end{bmatrix} \quad (9)$$

$$= \hat{A}_{i+x}^T \cdot \begin{bmatrix} I & \mathbf{0} \\ X & A_L \end{bmatrix}^{-1} \cdot \begin{bmatrix} \hat{v}_1^T \\ \vdots \\ \hat{v}_x^T \\ \hat{v}_{l_1+x}^T \\ \vdots \\ \hat{v}_{l_{k-1}+x}^T \end{bmatrix} \quad (10)$$

$$= \hat{A}_{i+x}^T \cdot \begin{bmatrix} I & \mathbf{0} \\ -A_L^{-1}X & A_L^{-1} \end{bmatrix} \cdot \begin{bmatrix} \hat{v}_1^T \\ \vdots \\ \hat{v}_x^T \\ \hat{v}_{l_1+x}^T \\ \vdots \\ \hat{v}_{l_{k-1}+x}^T \end{bmatrix}, \quad (11)$$

where we use X to denote the coefficients of components with v_i^T, $1 \le i \le x$.

By (4), we know that in PM-MSR codes, we only need to consider the coefficients of v_{l_i+x}, $1 \le i \le k-1$. From (11), we can know that such coefficients will be determined by $[\hat{a}_{i+x,1+x} \ \cdots \ \hat{a}_{i+x,k-1+x}] \cdot A_L^{-1} = A_i \cdot A_L^{-1}$.

Therefore, with L defined by the current helpers, we can have interference components that correspond with the $k-1$ elements in L, *i.e.*,

$$u_{i\to j}^T g_i F + h_j^T \sum_{l=1}^{k-1} a_{i,l} c_l = \sum_{m=1}^{k-1} a'_{i,m}(u_{l_m\to j}^T g_{l_m} F + h_j^T c_{l_m}) + \tilde{u}_{i\to j} g_j F. \quad (12)$$

Similarly, we can also rewrite (7) such that

$$\sum_{m=1}^{k-1} a'_{j',m}(u_{l_m\to j}^T g_{l_m} F + h_j^T c_{l_m}) + \tilde{u}_{j'\to j}^T g_j F = u_{j'\to j}^T g_{j'} F + h_j^T \sum_{m=1}^{k-1} a_{j',m} c_{l_m}. \quad (13)$$

Combining (8), (12), and (13), we know that with any t newcomers, we can reconstruct them after equivalently transforming Beehive codes with interference components tailored for these newcomers.

3.3 Implementation

So far we have explained how to construct Beehive codes and how to reconstruct multiple blocks with Beehive codes, from a mathematical perspective. In practice, we implement Beehive codes as linear codes, such that all operations can be performed as linear combinations on the given data. Given a generation of size $[k(d-k+1)+(k-1)(t-1)]w$ bytes, we fill all these bytes into an $(k(d-k+1)+(k-1)(t-1)) \times w$ matrix,

which can be partitioned as $[f_1^T \ c_1^T \ \cdots \ f_{k-1}^T \ c_{k-1}^T \ f_k^T]^T$. According to Sec. 3.2, the dimensions of $f_1, \ldots, f_k$ and $c_1, \ldots, c_{k-1}$ are $(d-k+1) \times w$ and $(t-1) \times w$, respectively.

With Beehive codes, the n blocks will be generated by matrix multiplication, *i.e.*, $G^B \cdot [f_1^T \ c_1^T \ \cdots f_{k-1}^T \ c_{k-1}^T \ f_k^T]^T$, where G^B is the generating matrix of Beehive codes. With the given system parameters, G^B should be an $n(d-k+t) \times (k(d-k+1)+(k-1)(t-1))$ matrix. Since every block encoded by Beehive codes contains $d-k+t$ segments, we can partition G^B into n submatrices, by every $(d-k+t)$ rows. In other words, we can write

$$G^B = \begin{bmatrix} g_1^B \\ \vdots \\ g_n^B \end{bmatrix},$$

where g_i^B is a matrix of size $(d-k+t) \times (k(d-k+1)+(k-1)(t-1))$, $1 \le i \le n$. Hence, the block i can be denoted as $g_i^B \cdot [f_1^T \ c_1^T \ \cdots \ f_{k-1}^T \ c_{k-1}^T \ f_k^T]^T$.

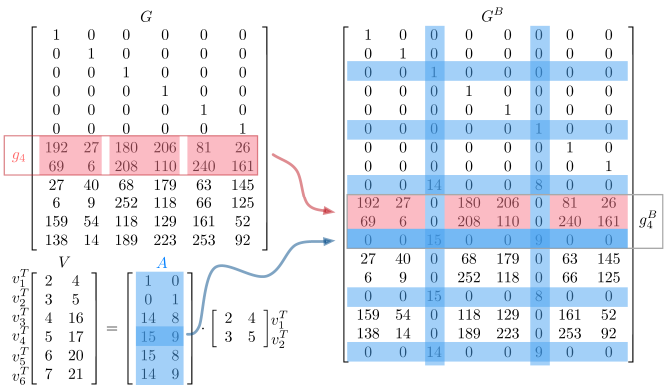


Fig. 6. An example of $(3,3,4,2)$ Beehive codes. This figure demonstrates the construction of the generating matrix (G^B) by combining the generating matrices of $(3,3,4)$ PM-MSR codes (G) and $(2,4)$ RS codes (A).

Assume that the generating matrices of PM-MSR codes and RS codes inside the Beehive codes are G and A. The block i will contain $g_i F$ and $\sum_{l=1}^{k-1} a_{i,l} c_l$. As illustrated in Fig. 6, we partition g_i into k submatrices of an equal size. Besides, we have $\sum_{l=1}^{k-1} a_{i,l} c_l = \sum_{l=1}^{k-1} a_{i,l} I_{t-1} c_l$, where I_{t-1} is a $(t-1) \times (t-1)$ identity matrix. Then we can construct g_i^B by interweaving parts of g_i and A_i alternatively. Notice that this way we can make G^B contain an identity matrix on top, and thus we can find the original data of F and C sequentially in the first k blocks, which helps users of storage systems to locate data easily and to read data sequentially.

During reconstruction, a helper i is going to calculate $[u_{i\to j}^T \ h_j^T] \cdot g_i^B [f_1^T \ c_1^T \ \cdots \ f_{k-1}^T \ c_{k-1}^T \ f_k^T]^T$, and send the result of the multiplication to the newcomer j. This is straightforward following our analysis above in Sec. 3.2.

We use $R_{i\to j}$, a row vector of size w bytes, to represent the segment transferred from block i to block j, where i and j can be either a helper or a newcomer. Further, we use $R_{D\to j}$ to denote all the segments sent from helpers to the newcomer j. For example, if $D = \{1,2\}$, $R_{D\to 3} = \begin{bmatrix} R_{1\to 3} \\ R_{2\to 3} \end{bmatrix}$. Similarly, we use $R_{j\to N}$ and $R_{N\to j}$ to denote the segments sent to the newcomer j from other newcomers and the segments sent from other newcomers to the newcomer j, respectively. Notice that at this time, the newcomer j should be excluded from the matrix. For example, if $N = \{3,4,5\}$, $R_{N:\to 4} = \begin{bmatrix} R_{3\to 4} \\ R_{5\to 4} \end{bmatrix}$.

We use $i_1, \ldots, l_d$ to denote the d helpers in D where $i_1 < \cdots < l_d$, and we let $L = \{i_1, \ldots, i_{k-1}\}$. Then we can have the corresponding A'_i from this L, $1 \le i \le n$ by (8).

Receiving d segments from helpers, the newcomer j will first calculate

$$\begin{bmatrix} (A'_{i_1})^T & \tilde{u}_{i_1\to j}^T \\ \vdots & \vdots \\ (A'_{i_d})^T & \tilde{u}_{i_d\to j}^T \end{bmatrix}^{-1} \cdot R_{D:\to j}. \tag{14}$$

By (12), the result of the equation above will be $\begin{bmatrix} u_{i_1\to j}^T g_{i_1} F + h_j^T c_{i_1} \\ \vdots \\ u_{i_{k-1}\to j}^T g_{i_{k-1}} F + h_j^T c_{i_{k-1}} \\ g_j F \end{bmatrix}$. In other words, we can find $g_j F$ in the last $d-k+1$ rows of the result.

Similarly, we use $j_1, \ldots, j_t$ to denote the t newcomers in N where $j_1 < \ldots < j_t$. Without loss of generality, we assume that $j = j_s$ where $1 \le s \le t$. To send data to other newcomers, the newcomer j will calculate

$$\begin{bmatrix} (A'_{j_1})^T & \tilde{u}_{j_1\to j}^T \\ \vdots & \vdots \\ (A'_{j_{s-1}})^T & \tilde{u}_{j-1\to j}^T \\ (A'_{j_{s+1}})^T & \tilde{u}_{j+1\to j}^T \\ \vdots & \vdots \\ (A'_{j_t})^T & \tilde{u}_{j_t\to j}^T \end{bmatrix} \cdot \begin{bmatrix} u_{i_1\to j}^T g_{i_1} F + h_j^T c_{i_1} \\ \vdots \\ u_{i_{k-1}\to j}^T g_{i_{k-1}} F + h_j^T c_{i_{k-1}} \\ g_j F \end{bmatrix}, \tag{15}$$

which is $R_{j\to N}$. In other words, the newcomer can send one row of the result to each of the other $t-1$ newcomers.

The newcomer j will also receive $t-1$ segments from other newcomers, *i.e.*, $R_{N\to j}$. By (13), we can get $\sum_{l=1}^{k-1} a'_{j,l} c_{i_l}$ by calculating

$$\begin{bmatrix} \tilde{u}_{j\to j_1}^T & h_{j_1}^T \\ \vdots & \vdots \\ \tilde{u}_{j\to j_{s-1}}^T & h_{j_{s-1}}^T \\ \tilde{u}_{j\to j_{s+1}}^T & h_{j_{s+1}}^T \\ \vdots & \vdots \\ \tilde{u}_{j\to j_t}^T & h_{j_t}^T \\ I_{d-k+1} & \mathbf{0} \end{bmatrix}^{-1} \cdot \begin{bmatrix} g_j F \\ R_{N\to j} \end{bmatrix}. \tag{16}$$

The first $t-1$ rows of the result will be $\sum_{l=1}^{k-1} a'_{j,l} c_{i_l} = \sum_{l=1}^{k-1} a_{j,l} c_l$.

4 MDS Beehive Codes

Beehive codes are not MDS codes, *i.e.*, not achieving the optimal storage overhead. The reason, from a mathematical perspective, is that in the construction C contains only $k-1$ blocks while we assume that any k blocks can recover the original data. In this section, we extend the construction of Beehive codes to make *MDS Beehive* codes, an MDS version of Beehive codes. The MDS Beehive codes, as the name suggests, achieve the optimal storage overhead. However,

a bit more traffic than Beehive codes will be incurred to reconstruct the same number of blocks, if blocks have the same size.

Since MDS Beehive codes is extended from the construction of Beehive codes, many parts of the construction is similar to Beehive codes, as well as the reconstruction operation. Therefore, we will focus on the differences between them in this section.

4.1 Code construction

MDS Beehive codes have the same system parameters as Beehive codes. To construct MDS Beehive codes, we still divide a generation of data into two parts, F and C, where both F and C will be divided into k blocks. To construct (k, r, d, t) MDS Beehive codes, we will encode F with $(k, r, d-1)$ PM-MSR codes, such that each block in F will have $d-k$ segments, instead of $d-k+1$ segments with Beehive codes. Because of this, the value of d of MDS Beehive codes should be no less than $2(k-1)+1 = 2k-1$. Each block in C will still have $t-1$ segments, which will be encoded by (k, r) RS codes. A block computed by MDS Beehive codes will then combine the $d-k$ segments from F and the $t-1$ segments from C. If each segment has w bytes, a generation with MDS Beehive codes will contain $k(d-k+t-1)w$ bytes.

We know that from the $(k, r, d-1)$ PM-MSR codes, we can get an associated $n \times (k-1)$ matrix A by letting $L = \{1, \ldots, k-1\}$. However, in order to encode C, we need to have a generating matrix with k columns, since this time we have k blocks in C. Therefore, we need to expand A with one more column to accommodate the new block while encoding the other $k-1$ blocks in the same way as Beehive codes.

We use A^M to denote the generating matrix to encode C of MDS Beehive codes. In other words, $A^M = [A \quad M]$ where M is a column vector with n bytes. It is required that after this expansion any k rows in A^M is linearly independent.

Remember that under PM-MSR codes, A is obtained from the matrix composed of vectors in $\hat{V}$, such that

$$\hat{V}^T = \begin{bmatrix} I_x & \mathbf{0} \\ X & A \end{bmatrix} \cdot \begin{bmatrix} \hat{v}_1^T \\ \vdots \\ \hat{v}_x^T \\ \hat{v}_{x+1}^T \\ \vdots \\ \hat{v}_{x+k-1}^T \end{bmatrix},$$

where we use X to denote coefficients of components with $\hat{v}_i$, $1 \le i \le x$, just like what we do in (10).

In the construction of PM-MSR codes, $\hat{V}^T$ is required that any $x+k-1$ rows are linearly independent. Here we choose $\hat{V}$ to be a Vandermonde matrix[4] to construct the PM-MSR codes because it is easy to be extended by adding one more column on the right. Then we add one more row at the bottom, such that the new matrix W can be written as

$$W^T = \begin{bmatrix} \hat{V}^T & y \\ \mathbf{0} & 1 \end{bmatrix}, \tag{17}$$

4. We can also choose the Cauchy matrix and it can be expanded in a very similar way.

where $y = (y_1 \ \cdots \ y_{n+x})^T$ is a column vector with $n+x$ bytes, used to expand $\hat{V}$ to be a larger Vandermonde matrix. From (17) it is easy to see that any k rows in W are linearly independent. We use w_i to denote the i-th column in W. If $1 \le i \le n+x$,

$$\begin{aligned} w_i &= \begin{bmatrix} \hat{v}_i \\ y_i \end{bmatrix} \\ &= \sum_{l=1}^{k+x-1} a_{i,l} \begin{bmatrix} \hat{v}_l \\ y_l \end{bmatrix} + (y_i - \sum_{l=1}^{k+x-1} a_{i,l} y_l) \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \\ &= \sum_{l=1}^{k+x-1} a_{i,l} w_l + (y_i - \sum_{l=1}^{k+x-1} a_{i,l} y_l) w_{n+x+1}. \end{aligned}$$

Therefore, we can have

$$\begin{bmatrix} w_1 \\ \vdots \\ w_{k+x-1} \\ w_{k+x} \\ \vdots \\ w_{n+x} \end{bmatrix} = \begin{bmatrix} I_x & \mathbf{0} \\ X & A^M \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ \vdots \\ w_{k+x-1} \\ w_{n+x+1} \end{bmatrix},$$

where $M = \begin{bmatrix} \mathbf{0} \\ y_{k+x} - \sum_{l=1}^{k+x-1} a_{k+x,l} y_l \\ \vdots \\ y_{n+x} - \sum_{l=1}^{k+x-1} a_{n+x,l} y_l \end{bmatrix}$. In this way, we can expand A into A^M while any k rows in A are linearly independent, which will be used as the generating matrix to encode C.

Assume G is the generating matrix of the $(k, r, d-1)$ PM-MSR codes, and let $M = [m_1 \ \ldots \ m_n]^T$. Then each block computed from MDS Beehive codes can be represented as $g_i F$ and $\sum_{i=1}^{k-1} a_{i,l} c_l + m_i c_k$, $1 \le i \le n$.

Inside MDS Beehive codes, the PM-MSR codes and the RS codes are both MDS codes, and thus MDS Beehive codes are also MDS. This can also be verified as any k blocks with MDS Beehive codes will contain the same amount of data as the original data, and they can recover the original data by solving F and C with the PM-MSR and the RS codes, respectively. Since MDS Beehive codes require that $d \ge 2k-1$, the highest code rate will be no more than $\frac{1}{2}$ for all values of t.

4.2 Reconstruction

The way of reconstruction of MDS Beehive codes is also extended from Beehive codes. For simplicity, we first discuss the case where no newcomer is in $\{1, \ldots, k-1\}$. Given a helper i and a newcomer j, the helper will send $u_{i\to j}^T g_i F + h_j^T (\sum_{l=1}^{k-1} a_{i,l} c_l + m_i c_k)$ to the newcomer, which can be written as

$$\begin{aligned} &u_{i\to j}^T g_i F + h_j^T (\sum_{l=1}^{k-1} a_{i,l} c_l + m_i c_k) = \\ &\sum_{l=1}^{k-1} a_{i,l} (u_{l\to j}^T g_l F + h_j^T c_l) + \tilde{u}_{i\to j} g_j F + m_i h_j^T c_k. \end{aligned} \tag{18}$$

With d helpers, the newcomer will be able to solve $g_j F$ (containing $d-k$ segments) and $h_j^T c_k$, as well as the $k-1$

interference components $a_{i,l}(u_{l\to j}^T g_l F + h_j^T c_l)$, $1 \le l \le k-1$. The newcomer j will then send the following data to the newcomer j':

$$\sum_{l=1}^{k-1} a_{j',l}(u_{l\to j}^T g_l F + h_j^T c_l) + m_{j'} h_j^T c_k + \tilde{u}_{j'\to j}^T g_j F$$
$$= u_{j'\to j}^T g_{j'} F + h_j^T (\sum_{l=1}^{k-1} a_{j',l} c_l + m_{j'} c_k).$$

Hence, the newcomer j' can cancel out the component with $g_{j'}F$ and then solve $\sum_{l=1}^{k-1} a_{j',l}c_l + m_{j'}c_k$ with the $t-1$ segments received from all other newcomers.

From the analysis in Sec. 3.2, we know that to discuss the reconstruction with general newcomers, we only need to consider A, which will be affected when we change L to a different set. Therefore, it is easy to see that (8)-(11) can still be applied with MDS Beehive codes. By adding $m_i h_j^T c_k$ and $m_{j'} h_j^T c_k$ at both sides of (12) and (13) respectively, we can also prove that any t blocks can be reconstructed with MDS Beehive codes.

4.3 Implementation

Similar to Beehive codes, with MDS Beehive codes we group the original data into generations of size $k(d-k+t-1)w$ bytes. Each generation will be filled into an $k(d-k+t-1)\times w$ matrix, which can be partitioned as $[f_1^T \ c_1^T \ \cdots f_k^T \ c_k^T]^T$. The dimensions of $f_1, \ldots, f_k$ and $c_1, \cdots, c_k$ are $(d-k)\times w$ and $(t-1)\times w$, respectively.

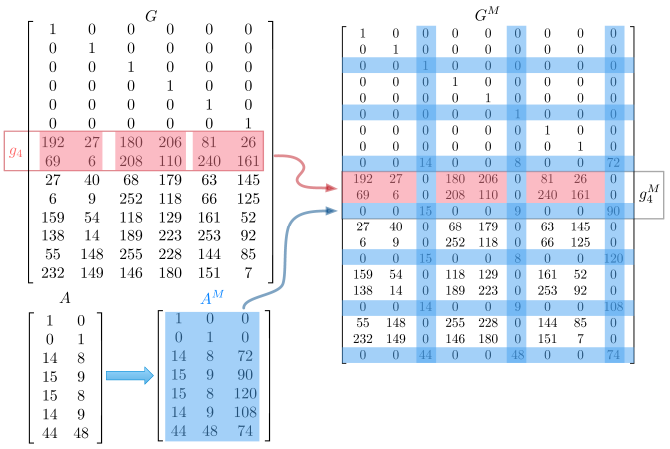


Fig. 7. An example of $(3,4,5,2)$ MDS Beehive codes. This figure demonstrates the construction of the generating matrix (G^M) by combining the generating matrix (G) of $(3,4,4)$ PM-MSR codes and A^M extended from the generating matrix (A) of $(3,4)$ RS codes.

We use Fig. 7 to illustrate how we construct the generating matrix G^M of MDS Beehive codes. The dimension of G^M is $n(d-k+t-1)\times k(d-k+t-1)$, where every $d-k+t-1$ rows can be used to compute one block. In other words, if we partition G^M as

$$G^M = \begin{bmatrix} g_1^M \\ \vdots \\ g_n^M \end{bmatrix},$$

where g_i^M is a $(d-k+t-1)\times k(d-k+t-1)$ matrix, $i = 1, \ldots, n$, the block i can be computed as $g_i^M \cdot [f_1^T \ c_1^T \ \cdots \ f_k^T \ c_K^T]$. In Fig. 7, we first expand A as described in Sec. 4.2, and then construct G^M by grouping elements in G and A^M alternatively like the construction of Beehive codes.

This construction gives us non-systematic codes, as we can see in Fig. 7. We can easily convert it equivalently into systematic codes, using a technique applied in [7] and [15]. Given a generating matrix G^M, we can convert it by calculating $G^M \cdot G_0^M$ where G_0^M contains the top $k(d-k+t-1)$ rows in G^M. Hence, we can find an identity matrix at the top of the result, making the corresponding codes systematic. The codes converted in this way can be reconstructed in the same way as the original codes.

TABLE 1
Comparison of different kinds of erasure codes for distributed storage systems.

	storage overhead	network transfer of reconstruction	disk I/O of reconstruction
RS	$\frac{n}{k}$	kt	kt
MSR	$\frac{n}{k}$	$\frac{d}{d-k+1}t$	dt
Beehive	$(\frac{n}{k}, \frac{n}{k-1})$	$\frac{d+t-1}{d-k+t}t$	d
MDS Beehive	$\frac{n}{k}$	$\frac{d+t-1}{d-k+t-1}t$	d
MSCR	$\frac{n}{k}$	$\frac{d+t-1}{d-k+t}t$	d

5 Discussions

5.1 Performance Analysis and Comparison

Now we compare the theoretical performance of Beehive and MDS Beehive, with other erasure codes for distributed storage systems including RS codes, MSR codes and MSCR codes (though the existing constructions of MSCR are not general). In the comparison, we assume that all blocks computed by all these codes have the same size, containing 1 unit of data. Among all $n = k + r$ blocks, any k blocks can recover the original data. For all codes, we compare the performance to reconstruct t blocks.

We summarize the results of the comparison in Table 1. We can see that all erasure codes in this table, except for Beehive codes, achieve the same optimal storage overhead, since they are all MDS codes. Beehive codes, on the other hand, increase the storage overhead to no more than $\frac{n}{k-1}$. Since in our comparison all n blocks contain n units of data, Beehive codes can store less original data than MDS codes.

Among all codes in this table, RS and MSR codes reconstruct blocks separately, and hence their network transfer and disk I/O consumed during reconstruction increases linearly with the number of blocks to reconstruct. MSCR codes define the optimal network transfer to reconstruct (k,r) MDS codes, and the disk I/O becomes irrelevant to the number of reconstructed blocks. We can see that Beehive codes and MDS Beehive codes coincide with the optimal network transfer and the optimal storage overhead of MSCR codes, respectively.

With the same system parameters, MDS Beehive codes incur $1 + \frac{1}{d-k+t-1}$ times network transfer of MSCR codes while achieving the same storage overhead, which can be made close to 1 arbitrarily with a large enough d or t. With

practical values of system parameters, this overhead will also be marginal.

5.2 Extensions

As we know, the construction of both Beehive codes and MDS Beehive codes is developed based on the property of interference alignment in (4). In fact, MSR codes that satisfy this property does not only include PM-MSR codes. For example, it can be proved that the construction proposed by Suh *et al.* [23] can satisfy this property as well. However, this construction requires that $d \geq 2k-1$, which is less general than the PM-MSR codes. Hence, in this paper our constructions are based on PM-MSR codes. The constructions with MSR codes in [23] can be in a very similar way.

Moreover, the constructions of Beehive and MDS Beehive codes can be extended by allowing newcomers to choose different d available blocks as helpers. By revisiting (6) and (18), we can find that any newcomer can solve $g_j F$ (and $h_j^T c_k$ with MDS Beehive codes), and the interference components. Then newcomers will exchange the same data even if they choose different helpers. Therefore, the reconstruction is no longer limited to d helpers and newcomers may connect to any d helpers, just like the MSCR codes proposed in [12]. However, more disk I/O can be increased when helpers chosen by different newcomers are not the same.

Though we mainly consider the case of $t > 1$ throughout the constructions of Beehive and MDS Beehive codes, we can notice that they can still be constructed when $t = 1$. At this time, Beehive codes will be equivalent as PM-MSR codes, as C will contain no data in Fig. 5. On the other hand, MDS Beehive codes will remain as MDS, but still incur slightly more network transfer during reconstruction.

The most desirable extension of Beehive and MDS Beehive codes, perhaps, is to combine their optimum together, *i.e.*, a construction of MSCR codes that achieve the optimal storage overhead and the optimal network transfer during reconstruction with a wide range of system parameters. We know that our constructions are quite close to this objective. However, it turns out non-trivial to merge this gap, and we leave this problem as our future work.

6 Evaluation

6.1 Evaluation settings

In our evaluation, we implement Beehive and MDS Beehive codes in C++, following the discussions of the implementation in Sec. 3.3 and Sec. 4.3. Hence, all operations, including encoding, decoding, and reconstruction, are implemented as vector/matrix multiplications on a finite field of size 2^8. We use the Intel storage acceleration library (ISA-L) [38] for the finite field arithmetic. Beside Beehive and MDS Beehive codes, we also implement RS and MSR codes with ISA-L, for comparison purposes.

All evaluations are performed on Amazon EC2 instances of type c3.xlarge, where we run operations of various erasure codes including RS, MSR, Beehive, and MDS Beehive codes. All operations are running with one single thread. We repeat each operation for 100 times, and plot the average results.

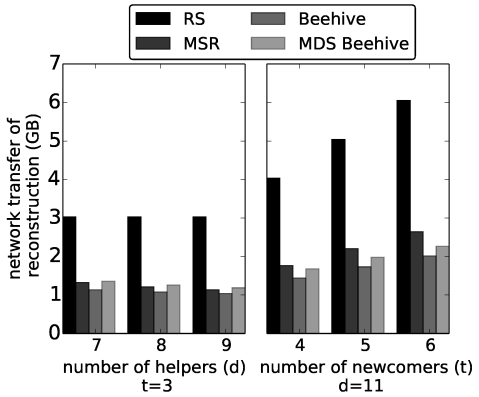


Fig. 8. The total amount of network transfer to reconstruct t newcomers, with d helpers (k helpers with RS codes).

By default, we set the block size to be 63 MB (except in Sec. 6.8). We choose this size to make sure that the block can always contain an integer number of segments with different values of system parameters, encoded with different erasure codes in our evaluations. Given the block size and the value of k, we will generate one generation of the original data randomly for corresponding erasure codes, in order to make sure that all blocks with different erasure codes will have the same size.

6.2 Network transfer

In Fig. 8, we compare the amount of network transfer incurred by the reconstruction with t newcomers. With RS codes and MSR codes, the t newcomers are reconstructed separately, by downloading data from k and d helpers, respectively. In all of our evaluations, we set $k = 4$, $t = 3$, $d = 7$, and $r = d + t - k$ by default, and change the values of d and t respectively except in Sec. 6.8. We can observe that Beehive codes achieve the minimum network transfer, saving up to 23.8% of network transfer of MSR codes, and up to 55.6% of RS codes. The saving of network transfer becomes more significant with more newcomers. With an increase of d, the network transfer can be slightly decreased as well. MDS Beehive codes demonstrate similar network transfer as Beehive codes. As expected, we can see that MDS Beehive codes incur more network transfer than Beehive codes. When $t = 3$, the network transfer of MDS Beehive codes and MSR codes are roughly the same. With more than 3 newcomers to reconstruct, MDS Beehive codes will also save more and more network transfer than MSR codes.

6.3 Disk I/O

Fig. 9 compares the total amount of disk read on helpers to reconstruct t newcomers, where we use (MDS) Beehive to represent both Beehive and MDS Beehive codes as they consume the same amount of disk I/O on helpers. It is easy to understand that (MDS) Beehive codes incur much less disk read than both RS (by up to 70.8%) and MSR codes (by up to 83.3%), because data reads, which will be incurred multiple times with RS or MSR codes, can be coalesced to be only once.

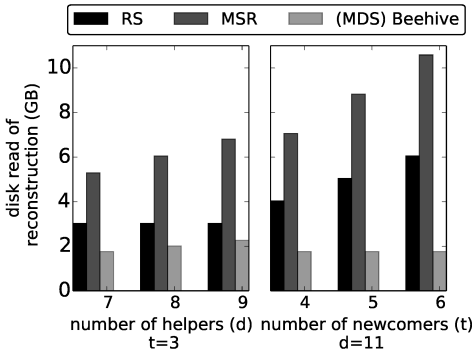


Fig. 9. The total amount of disk read on helpers to reconstruct t newcomers.

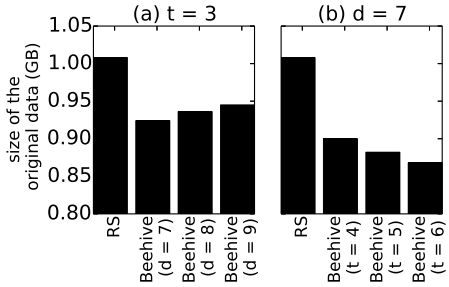


Fig. 10. The size of the original data in a generation that can be encoded into k blocks of size 63 MB.

6.4 Storage efficiency

As mentioned before, in our evaluations we assume that all blocks with different erasure codes have the same size, for the purpose of a fair comparison of all operations. Since Beehive codes are not MDS codes, it can be expected that the original data in a generation with Beehive codes will be less than MDS codes.

In Fig. 10 we compare the size of the original data in a generation with RS codes and Beehive codes. MSR and MDS Beehive codes will have the same size of the original data as RS codes since they are all MDS codes. With $k = 4$, we can compute by (5) that Beehive codes can store at most 25% less original data than MDS codes. In Fig. 10, we can observe that the original data with Beehive codes is up to 13.9% less than RS codes. By (5), we know that the original data with Beehive codes contain $t-1$ fewer segments than MDS codes. With a higher number of d, the total number of segments in the original data will also increase, and hence the loss of storage efficiency will be decreased. With more newcomers to reconstruct, however, we can observe less original data with Beehive codes since the gap will increase as well.

6.5 Completion time of reconstruction

The completion time to compute the data for reconstruction is shown in Fig. 11 and Fig. 12. Different erasure codes compute data differently during reconstruction. For example, RS codes only need to compute the data at newcomers, whereas MSR codes also require helpers to compute data for the newcomer. With Beehive codes and MDS Beehive codes, the operation at newcomers will be further divided into two stages, computing the data for other newcomers and the

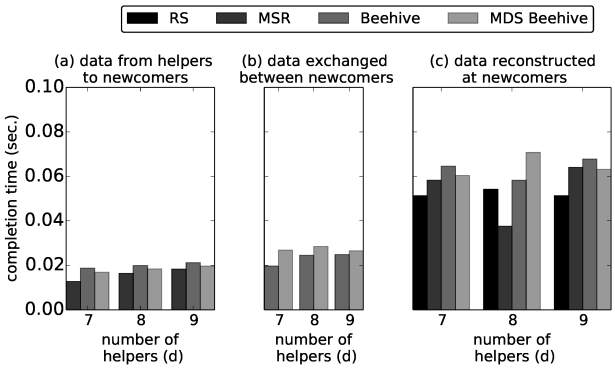


Fig. 11. Completion time of the reconstruction, with different numbers of helpers

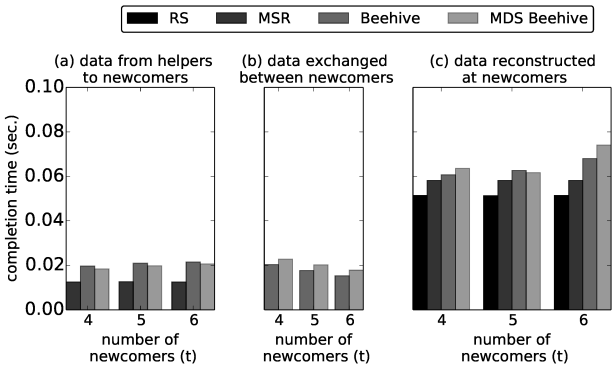


Fig. 12. Completion time of reconstruction, with different numbers of newcomers

data we need to reconstruct. From Fig. 11 and Fig. 12, we can see that Beehive codes and MDS Beehive codes achieve similar performance as RS and MSR codes, with different values of d and t.

6.6 Encoding performance

We measure the time to encode one generation of the original data into $k + r$ blocks with various erasure codes. In Fig. 13, we find that Beehive and MDS Beehive codes have similar performance of encoding, and their completion time is slower than RS and MSR codes. We believe that this is because both Beehive and MDS Beehive codes are constructed by combining RS and MSR codes, increasing the complexity to encode data. Intuitively, since MSR codes have higher completion time of encoding with a large d, implying that the encoding complexity increases with the number of segments in a block, Beehive and MDS Beehive codes will also have encoding complexity increasing with d and t as the number of segments inside a block increases with d and t as well, which we can also observe from Fig. 13.

6.7 Decoding performance

In the evaluation, we manually remove t data blocks and recover the original data from k blocks, including the $k - t$ remaining data blocks and t parity blocks. Similar to encoding, we observe similar performance in the decoding as Beehive and MDS Beehive codes spend more time to decode the original data in a generation, as shown in Fig. 14. The

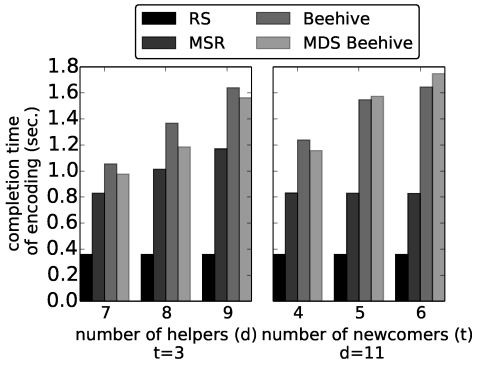


Fig. 13. Completion time of encoding.

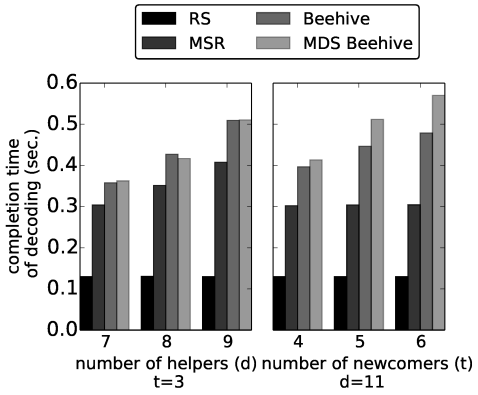


Fig. 14. Completion time of decoding the original data from $k - t$ data blocks and t parity blocks.

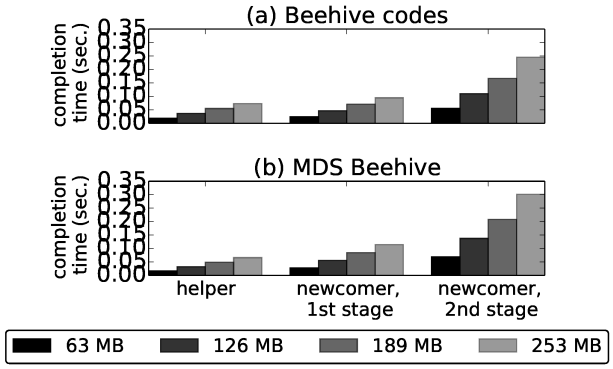


Fig. 15. Completion time of reconstruction with different block sizes, with $k = 4, d = 7, t = 3$, and $r = 6$.

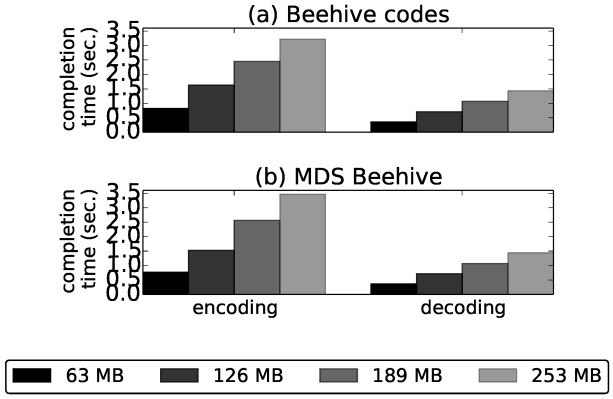


Fig. 16. Completion time of encoding and decoding with different block sizes with $k = 4, d = 7, t = 3$, and $r = 6$.

performance of Beehive and MDS Beehive codes suggests that they are more suitable for archival storage with less data access while the performance of the reconstruction is of more importance.

6.8 Scalability

Fig. 15 and Fig. 16 show that the time of all operations with Beehive codes and MDS Beehive codes linearly increase with the size of the block. Hence, we can say that Beehive codes and MDS Beehive codes can scale very well with large files. The reason is that all operations with Beehive and MDS Beehive codes are performed by vector/matrix multiplications, and with more data we will only have the matrix on the right (*e.g.*, in (14)-(16)) with more columns.

7 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. We further extend the construction of Beehive codes to construct an MDS version of Beehive codes that achieve the optimal storage overhead with marginally more network transfer during reconstruction. Through evaluations on Amazon EC2, we show that Beehive codes and MDS Beehive codes can both save network transfer and disk I/O significantly, compared to existing erasure codes like RS and MSR codes.

Acknowlegments

We would like to thank our editor and anonymous reviewers for their efforts on improving the quality and the presentation of this paper. This paper is partially supported by the NSERC Discovery research program and the SAVI NSERC Strategic Networks grant.

References

[1] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing Elephants: Novel Erasure Codes for Big Data," *Proc. VLDB Endowment*, 2013.

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

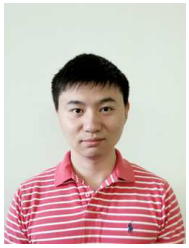
[3] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure Coding in Windows Azure Storage," in *Proc. USENIX Annual Technical Conference (USENIX ATC)*, 2012.

[4] Z. Zhang, A. Wang, K. Zheng, G. Uma Maheswara, and B. Vinayakumar, "Introduction to HDFS Erasure Coding in Apache Hadoop," http://blog.cloudera.com/blog/2015/09/introduction-to-hdfs-erasure-coding-in-apache-hadoop/.

[5] K. Bandaru and K. Patiejunas, "Under the Hood: Facebook's Cold Storage System," https://code.facebook.com/posts/1433093613662262/-under-the-hood-facebook-s-cold-storage-system-/.

[6] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "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.

[7] 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," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5227–5239, 2011.
[8] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in Globally Distributed File Systems," in *Proc. USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2010.
[9] A. Ma, F. Douglis, G. Lu, D. Sawyer, S. Chandra, and W. Hsu, "RAIDShield: Characterizing, Monitoring, and Proactively Protecting Against Disk Failures," in *Proceedings of the 13th USENIX conference on File and Storage Technologies*, 2015.
[10] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker, "Total Recall: System Support for Automated Availability Management," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.
[11] Y. Hu, Y. Xu, X. Wang, C. Zhan, and P. Li, "Cooperative Recovery of Distributed Storage Systems from Multiple Losses with Network Coding," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 2, pp. 268–276, 2010.
[12] K. W. Shum and Y. Hu, "Cooperative Regenerating Codes for Distributed Storage Systems," *IEEE Transactions on Information Theory*, vol. 59, no. 11, pp. 7229–7258, 2013.
[13] J. Li and B. Li, "Cooperative Repair with Minimum-Storage Regenerating Codes for Distributed Storage," in *Proc. IEEE INFOCOM*, 2014.
[14] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network Coding for Distributed Storage Systems," *IEEE Trans. Inform. Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.
[15] K. V. Rashmi, P. Nakkiran, J. Wang, N. B. Shah, and K. Ramchandran, "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.
[16] J. S. Plank and L. Xu, "Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications," in *Proc. IEEE International Symposium on Network Computing and Applications (NCA)*, 2006.
[17] V. R. Cadambe and S. A. Jafar, "Interference Alignment and Degrees of Freedom of the K-User Interference Channel," *IEEE Transactions on Information Theory*, vol. 54, no. 8, pp. 3425–3441, 2008.
[18] A. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A Survey on Network Codes for Distributed Storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, Mar. 2011.
[19] Y. Hu, P. P. C. Lee, and K. W. Shum, "Analysis and Construction of Functional Regenerating Codes with Uncoded Repair for Distributed Storage Systems," in *Proc. IEEE INFOCOM*, 2013.
[20] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Explicit Codes Minimizing Repair Bandwidth for Distributed Storage," in *IEEE Information Theory Workshop on Information Theory (ITW)*, 2010, pp. 1–5.
[21] S. Goparaju, A. Fazeli, and A. Vardy, "Minimum Storage Regenerating Codes for All Parameters," *arXiv preprint arXiv:1602.04496*, 2016.
[22] N. Shah, K. V. Rashmi, P. Kumar, and K. Ramchandran, "Interference Alignment in Regenerating Codes for Distributed Storage: Necessity and Code Constructions," *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2134–2158, 2012.
[23] C. Suh and K. Ramchandran, "Exact-Repair MDS Code Construction Using Interference Alignment," *IEEE Trans. Inf. Theory*, vol. 57, no. 3, pp. 1425–1442, Mar. 2011.
[24] V. R. Cadambe, S. A. Jafar, H. Maleki, K. Ramchandran, and C. Suh, "Asymptotic Interference Alignment for Optimal Repair of MDS Codes in Distributed Storage," *IEEE Transactions on Information Theory*, vol. 59, no. 5, pp. 2974–2987, 2013.
[25] Z. Wang, I. Tamo, and J. Bruck, "On Codes for Optimal Rebuilding Access," in *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*. IEEE, 2011, pp. 1374–1381.
[26] I. Tamo, Z. Wang, and J. Bruck, "Zigzag Codes: MDS Array Codes With Optimal Rebuilding," *IEEE Transactions on Information Theory*, vol. 59, no. 3, pp. 1597–1616, 2013.
[27] E. En Gad, R. Mateescu, F. Blagojevic, C. Guyot, and Z. Bandic, "Repair-Optimal MDS Array Codes Over GF(2)," in *Proc. IEEE International Symposium on Information Theory Proceedings (ISIT)*, 2013, pp. 887–891.
[28] Y. Wang, X. Yin, and X. Wang, "MDR Codes: A New Class of RAID-6 Codes With Optimal Rebuilding and Encoding," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 1008–1018, May 2014.
[29] ——, "Two New Classes of Two-Parity MDS Array Codes With Optimal Repair," *IEEE Communications Letters*, vol. 20, no. 7, pp. 1293–1296, 2016.
[30] B. Sasidharan, G. K. Agarwal, and P. V. Kumar, "A High-Rate MSR Code With Polynomial Sub-Packetization Level," in *2015 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2015, pp. 2051–2055.
[31] A. S. Rawat, O. O. Koyluoglu, and S. Vishwanath, "Progress on High-Rate MSR Codes: Enabling Arbitrary Number of Helper Nodes," *arXiv preprint arXiv:1601.06362*, 2016.
[32] B. Sasidharan, M. Vajha, and P. V. Kumar, "An Explicit, Coupled-Layer Construction of a High-Rate MSR Code With Low Sub-Packetization Level, Small Field Size and All-Node Repair," *arXiv preprint arXiv:1607.07335*, 2016.
[33] A.-M. Kermarrec, N. Le Scouarnec, and G. Straub, "Repairing Multiple Failures with Coordinated and Adaptive Regenerating Codes," *IEEE International Symposium on Network Coding (NetCod)*, 2011.
[34] A. Wang and Z. Zhang, "Exact Cooperative Regenerating Codes with Minimum-Repair-Bandwidth for Distributed Storage," in *Proc. IEEE INFOCOM*, 2013.
[35] J. Chen and K. W. Shum, "Repairing Multiple Failures in the Suh-Ramchandran Regenerating Codes," in *Proc. IEEE Int. Symp. Inform. Theory (ISIT)*, 2013.
[36] N. Le Scouarnec, "Exact Scalar Minimum Storage Coordinated Regenerating Codes," *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2012.
[37] K. W. Shum and J. Chen, "Cooperative Repair of Multiple Node Failures in Distributed Storage Systems," *arXiv preprint arXiv:1607.08322*, 2016.
[38] "Intel Storage Acceleration Library," https://01.org/intel%C2%AE-storage-acceleration-library-open-source-version.



Jun Li received his B.S. and M.S. degrees from the School of Computer Science, Fudan University, China, in 2009 and 2012. He is currently with the Department of Electrical and Computer Engineering at the University of Toronto, working towards his Ph.D. degree. His research interest focuses on large-scale distributed storage systems with erasure coding.

Baochun Li received his Ph.D. degree from the Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, in 2000. Since then, he has been with the Department of Electrical and Computer Engineering at the University of Toronto, where he is currently a Professor. He holds the Bell Canada Endowed Chair in Computer Engineering since August 2005. His research interests include large-scale distributed systems, cloud computing, peer-to-peer networks, applications of network coding, and wireless networks. He is a member of the ACM and a fellow of the IEEE.