A Non-MDS Erasure Code Scheme for Storage Applications

A. Kiani, S. Akhlaghi
Shahed University, Faculty of engineering
kiani@shahed.ac.ir, akhlaghi@shahed.ac.ir
Corresponding author: S. Akhlaghi

***Abstract*— This paper investigates the use of redundancy and self-repairing against node failures in distributed storage systems using a novel non-MDS erasure code. In replication method, access to one replication node is adequate to reconstruct a lost node, while in MDS erasure coded systems which are optimal in terms of redundancy-reliability tradeoff, a single node failure is repaired after recovering the entire stored data, thereby consuming more repair bandwidth. The current paper aims at investigating a new type of erasure codes with a reduced repair bandwidth as compared to conventional MDS erasure codes. Specifically, we propose a non-MDS $(2k, k)$ code that tolerates any three node failures and more importantly, it is shown using the proposed code a single node failure can be repaired through connecting to only three nodes which gives the ability to reduce the repair bandwidth comparing to MDS codes**

***Index Terms*—** Distributed storage systems, Erasure code, MDS code.

I. INTRODUCTION

The field of large scale data storage systems has witnessed significant growth in recent years with applications such as social networks and file sharing. In storage systems, data should be stored over multiple independent nodes, i.e., disks, servers, peers, etc. In such systems, it may happen that a storage node has failed or leaves the system unexpectedly. In this case, it is widely recognized that the use of redundancy information can maintain a reliable storage capability over individually unreliable nodes.

There are various strategies for distributing redundancy in which depending on the used method the system can tolerate a limited number of node failures. Moreover, the system should have the capability of self-repairing to keep the functionality of the system against node failures. To this end, each damaged node is replaced with a new node. Then, this newcomer should be connected to the existing nodes to download the same amount of data as the damaged node. Reconstructing a failed node and the maintenance bandwidth are called repair problem and repair bandwidth, respectively.

Manuscript Received 10-May-2013 and Revised on 11-Auguest-2013 *ISSN: 2322-3936*
Accepted on 20- September-2013

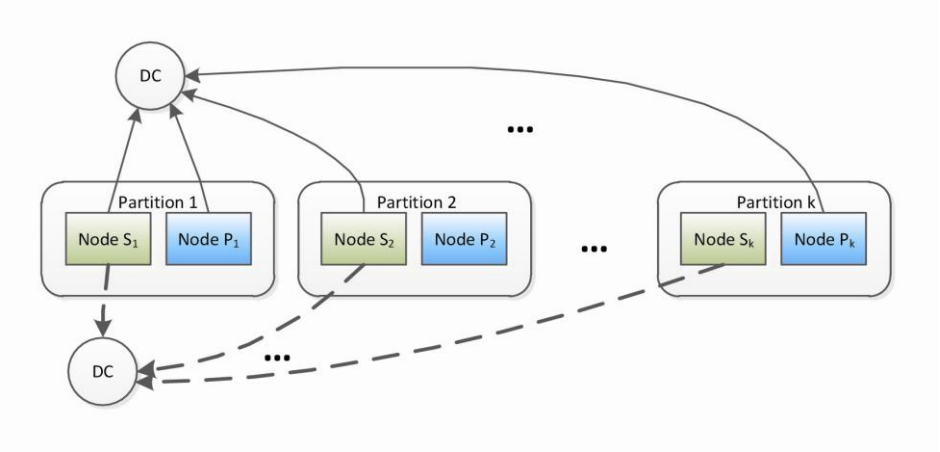

Fig. 1. The graphical representation of the proposed code. The recovery of original data file can be achieved by connecting to: (i) two nodes within a partition and $k-2$ different nodes selected over $k-2$ different partitions (solid-lines) or (ii) $2m \leq k$ parity nodes and $k-2m$ systematic nodes selected from k different partitions (dashed-lines are the specific case of $m=0$).

Erasure codes are the most common strategy for distributing redundancy across the network. An erasure coded system employs totally n packets of the same size, k of which are data packets (the fragments of the original data file) and $n-k$ of which are parity packets (the parity information). It is worth mentioning that the process of coding can be done using either Maximum Distance Separable (MDS) or non-MDS codes. In a distributed storage system, these packets are stored at n different nodes across the network. MDS codes [1] are optimally space-efficient and the encoding process is such that having access to any k nodes is adequate to recover the original data file. In these codes, each parity node increases the fault tolerance. In other words, a (n, k) MDS coded system can tolerate any $n-k$ node failures.

Replication, RAID (Redundant Array of Independent Disks) 5, RAID 6 [2], and Reed-Solomon codes [3] are the most popular MDS codes that have been used in storage systems. In replication, the parity nodes and data nodes are the same. In other words, a replication of each data node is stored in a related parity node. RAID 5 and RAID 6 employ $n-k=1$ and $n-k=2$ parity node(s), respectively. However, Reed-Solomon codes can be designed for any value of (n, k) MDS codes [3]. Another class of MDS codes are MDS array codes such as EVENODD codes [4]. These codes are based on XOR operation and have lower encoding and decoding complexity than conventional Reed-Solomon codes.

Non-MDS codes are introduced to reduce the computational complexity of encoding and decoding processes over lossy networks; however, are not as space-efficient as MDS codes. These codes are investigated in several papers. As a case in point, Hafner in [5] proposes a new class of non-MDS XOR-based codes, called WEAVER codes. The WEAVER codes are vertical codes which can tolerate up to 12 node failures. In a vertical code like X-code and WEAVER code each node contains both data and parity packets. In contrast, each node in a flat-XOR code such as EVENODD, holds

either data or parity packets. Greenan et al. In [6] describe construction of two novel flat XOR-based codes, called stepped combination and HD-combination codes.

The standard MDS codes are inefficient in terms of repair bandwidth as reconstructing a failed node consumes a repair bandwidth equal to the entire stored data. This motivated Dimakis et al. In [7] to propose a new type of codes, called regenerating codes (RC), which basically make a balance between the repair bandwidth and the storage capacity per node. The repair model presented in [7] is a functional repair. In the functional repair model the recreated packets stored at the replaced node can be different with the lost packets. This is in contrast to the exact repair in which each lost packet is exactly reconstructed. The exact repair problem for RCs is investigated in [8]. Also other variants of RCs are introduced in [9], [10].

Regenerating codes outperform existing MDS erasure codes in terms of maintenance bandwidth, however, constructing a new packet requires communication with $d \geq k$ nodes and the minimum repair bandwidth can be achieved when $d = n - 1$. In addition, the surviving nodes have to apply a random linear network coding to their stored packets. Accordingly, many of the proposed constructions require a huge finite-field size which are not feasible for practical storage systems. The current study aims to introduce a $(n, k) = (2k, k)$ non-MDS XOR-based code which can tolerate any three node failures. Accordingly, it is shown a single node failure can be exactly repaired through access to only three nodes regardless of k.

The rest of paper is organized as follows: Section II gives the construction steps following the main idea behind the proposed code. In Section III, we explain the repair problem of the proposed code. Finally, Section IV concludes the paper.

II. CONSTRUCTION

In this section we describe the construction of the proposed non-MDS code. Fig. (1) shows a graphical representation of this code. This code is a class of flat XOR-codes which contains $2k$ storage nodes where each node stores one packet. The construction is such that k out of $2k$ existing nodes i.e., $\{S_i\}_{i=1,\dots,k}$, hold data fragments, called systematic nodes. The remaining k nodes, i.e.,$\{P_i\}_{i=1,\dots,k}$, are the parity nodes which store parity packets. Also it is assumed that each systematic node (S_i) has a related parity node (P_i) in which they form a same partition. Thus, with this construction, the code entails k partitions.

For storing a file of size M using this construction, the file is divided into k fragments i.e., $d_1, d_2, d_3, \dots, d_k$, each of size $\frac{M}{k}$. The data stored in each parity node of a given partition is simply the XOR of data fragments from other $k - 1$ partitions.

Each fragment can be a single bit or a block of bits. These fragments are stored at k systematic nodes. Fig. (2) illustrates an $(n, k) = (10,5)$ code corresponding to the explained construction.

Referring to Fig. (2), the five data fragments, i.e., d_1, d_2, d_3, d_4 and d_5, are stored at nodes S_1, S_2, S_3, S_4 and S_5 respectively. Also, the parity packet p_i to be stored in parity node P_i is computed as

$$p_i = \sum_{j=1, j\neq i}^{k} d_i \quad for\, i = 1, \dots, k\,, \tag{1}$$

where addition here is bit-by-bit XOR of two data packets. For instance in a (10,5) code, as can be inferred from Fig. (2), parity packets $p_1 = d_2 + d_3 + d_4 + d_5$, $p_2 = d_1 + d_3 + d_4 + d_5$, $p_3 = d_1 + d_2 + d_4 + d_5$, $p_4 = d_1 + d_2 + d_3 + d_5$ and $p_5 = d_1 + d_2 + d_3 + d_4$ are stored in parity nodes P_1, P_2, P_3, P_4 and P_5 respectively. It is worth mentioning that for the specific case of $k = 2$ this code is same as the replication code. Also for $k = 3$, the parity packets are same with the parity packets of the proposed chain code in [6].

Now we are ready to discuss how the recovery of the original file can be made. It is assumed corresponding to a request to reconstructing the original data file a Data Collector (DC) is initiated and connects to existing nodes. Since, each node stores a data size $\frac{M}{k}$, thus a DC needs to connect to at least k out of existing nodes to reconstruct the original data file of size M. Recall that the proposed construction in this work does not have the MDS property. As a result, having access to any k nodes out of existing $2k$ nodes does not ensure restoring the original file. However, we will argue that a DC can wisely select k out of existing $n = 2k$ nodes to construct the whole data file. To this end, DC has actually two possible choices for selecting k storage nodes to connect to. First, a DC can connect to a systematic node and parity node of the same partition and select $k-2$ different nodes from $k-2$ different partitions out of the remaining $k-1$ partitions (solid-lines in Fig. (1) are a special case of this scenario). Noting there are $\binom{k}{k-1}$ options for selecting $k-1$ out of existing k partitions, $\binom{k-1}{1}$ options to select a systematic-parity pair from $k-1$ selected partitions, and 2^{k-2} options for selecting either of systematic or parity nodes of the remaining $k-2$ partitions, thus a DC has totaly $\binom{k}{k-1}\binom{k}{1}2^{k-2} = (k)(k-1)2^{k-2}$ options to choose k nodes to connect to. In the second scenario, a DC can connect to $2m \leq k$ parity nodes and $k-2m$ systematic nodes selected from k different partitions (dashed-lines in Fig. (1) can be considered as a special case of this scenario when $m = 0$). In this case, the number of ways to choose k nodes is computed as $\sum_{m=0}^{\lfloor \frac{k}{2} \rfloor}\binom{k}{2m} = 2^{k-1}$. Thus, adding the possible choices of two scenarios, there are totally $2^{k-2}(k^2 - k + 2)$ ways to recover the original file using k nodes. Considering the two possible scenarios, we need to have at least $k-1$ available partitions (having access to either of systematic or parity node is adequate for the remaining partitions) to reconstruct the original data file. Since, each systematic node and its related parity node constitutes a partition, the proposed $(2k, k)$ code can tolerate up to three node failures in general.

However, this code can tolerate up to $k-1$ node failures as long as the failed nodes are from $k-1$ different partitions.

As is mentioned earlier, the storage capacity per node for storing a file of size M is $\frac{M}{k}$ which is similar to the storage capacity of a standard MDS code as well as that of the Minimum Storage Regenerating (MSR) codes[1]. Although MSR and conventional MDS codes offer higher degrees of freedom for reconstructing the original data file, we will show that the repair bandwidth of the proposed method outperforms the aforementioned coding strategies. Recall that to keep the reliability of network across time, each failed node should be repaired. In MDS codes, a failure is fixed after transferring the whole data file over the network (the repair bandwidth is equal to M and the repair problem is done through connecting the new node to $d=k$ existing nodes). On the other hand, regenerating codes can further reduce the repair bandwidth if we allow the new node to connect to $d>k$ nodes. Our proposed code, however, has the ability to do this through connecting to $d<k$ nodes, which is an advantage as compared to regenerating codes. In other words, fewer nodes are involved through the course of downloading. The following section aims at addressing the repair model of the suggested code.

III. REPAIR PROBLEM

Note that when a node fails or leaves the system a new node is initiated, attempting to connect to existing nodes to reconstruct the failed node (exact repair problem). In this case, two scenarios may occur: (i) The parity or systematic node in the same partition as the damaged node (the neighboring node) is active, or (ii) The neighboring node has failed. In the presence of neighboring node, the failed node can be reconstructed thorough communicating to only three nodes i.e., the neighboring node and the parity and the systematic nodes from another active partition. In fact, there are $k-1$ ways to repair a failed node through downloading from the aforementioned three nodes. For example, referring to Fig. (2), we assume that the systematic node S_1 which holds data fragment d_1 is failed. When parity node P_1 which stores parity packet $p_1=d_2+d_3+d_4+d_5$ is active, the new node can restore $s_1=d_1$ through downloading three packets in one of the following $k-1=4$ ways,

$$s_1=(d_2+d_3+d_4+d_5)+d_2+(d_1+d_3+d_4+d_5)$$
$$(d_2+d_3+d_4+d_5)+d_3+(d_1+d_2+d_4+d_5)$$
$$(d_2+d_3+d_4+d_5)+d_4+(d_1+d_2+d_3+d_5)$$
$$(d_2+d_3+d_4+d_5)+d_5+(d_1+d_2+d_3+d_4)$$

1 The identified tradeoff curve in [7] has two extremal points; one end of this curve corresponds to the minimum storage per node and the other end corresponds to minimum bandwidth point. These two extremal points can be achieved by the use of the MSR and Minimum Bandwidth Regenerating (MBR) codes, respectively.

which leads to have a repair bandwidth equal to $3\frac{M}{k}$. As discussed earlier, in MSR codes the new node should connect $d \geq k$ nodes to ensure reconstructing a failed node. In these codes the repair bandwidth is computed as $\frac{Md}{k(d-k+1)}$ which is a decreasing function with respect to d [7] and, hence, when new node connects to the minimum possible nodes, i.e., k nodes, the repair bandwidth takes its maximum value, that is M. For instance, in a $\mathrm{MSR}(n=10, k=5)$ code, the repair bandwidth $3\frac{M}{5}$ can be achieved if new node connects to $d=6$ nodes which are greater than $d=3$ nodes in the proposed scheme.

In the second scenario, when the neighboring node is not available, one of the following strategies can be used to reconstruct the failed node. In the first strategy, dubbed strategy A, the parity node is first repaired and then using it the neighboring systematic node is repaired. In this case, the new parity node should connect to $2m$ parity nodes and $k-1-2m$ systematic nodes from $k-1$ different partitions. This results in $\sum_{m=0}^{\left\lfloor \frac{k-1}{2} \right\rfloor} \binom{k-1}{2m} = 2^{k-2}$ ways to choose $k-1$ nodes to connect to. For instance, referring to Fig. (2), for a $(n,k)=(10,5)$ non-MDS code there are $2^{(5-2)}=8$ ways in which the new node can use 0, 2 or 4 parity nodes to repair parity node P_1 which stores $p_1 = d_2 + d_3 + d_4 + d_5$ without the use of node S_1. These eight ways are as follows,

$$p_1 = (d_2) + (d_3) + (d_4) + (d_5)$$
$$= (d_1 + d_3 + d_4 + d_5) + (d_1 + d_2 + d_4 + d_5) + (d_4) + (d_5)$$
$$= (d_1 + d_3 + d_4 + d_5) + (d_1 + d_2 + d_3 + d_5) + (d_3) + (d_5)$$
$$= (d_1 + d_3 + d_4 + d_5) + (d_1 + d_2 + d_3 + d_4) + (d_3) + (d_4)$$
$$= (d_1 + d_2 + d_4 + d_5) + (d_1 + d_2 + d_3 + d_5) + (d_2) + (d_5)$$
$$= (d_1 + d_2 + d_4 + d_5) + (d_1 + d_2 + d_3 + d_4) + (d_2) + (d_4)$$
$$= (d_1 + d_2 + d_3 + d_5) + (d_1 + d_2 + d_3 + d_4) + (d_2) + (d_3)$$
$$= (d_1 + d_3 + d_4 + d_5) + (d_1 + d_2 + d_4 + d_5) +$$
$$(d_1 + d_2 + d_3 + d_5) + (d_1 + d_2 + d_3 + d_4)$$

In the second strategy, called strategy B, the failed systematic node is first repaired and then is used in the reconstruction of the neighboring parity node. To this end, the new systematic node requires to communicate with $2m-1$ parity nodes and $k-2m$ systematic nodes from $k-1$ different partitions, thus the number of possible ways to do this can be computed as $\sum_{m=1}^{\left\lceil \frac{k-1}{2} \right\rceil} \binom{k-1}{2m-1} = 2^{k-2}$. For example, as is shown in Fig. (2), there exist $2^{(5-2)}=8$ options for the new node to choose one or three parity nodes for repairing the systematic node S_1 which holds $s_1 = d_1$ when P_1 is simultaneously failed and is not accessible. These options are as follows

$$s_1 = (d_1 + d_3 + d_4 + d_5) + (d_3) + (d_4) + (d_5)$$

$$= (d_1 + d_2 + d_4 + d_5) + (d_2) + (d_4) + (d_5)$$
$$= (d_1 + d_2 + d_3 + d_5) + (d_2) + (d_3) + (d_5)$$
$$= (d_1 + d_2 + d_3 + d_4) + (d_2) + (d_3) + (d_4)$$
$$= (d_1 + d_3 + d_4 + d_5) + (d_1 + d_2 + d_4 + d_5) + (d_1 + d_2 + d_3 + d_5) + (d_5)$$
$$= (d_1 + d_3 + d_4 + d_5) + (d_1 + d_2 + d_4 + d_5) + (d_1 + d_2 + d_3 + d_4) + (d_4)$$
$$= (d_1 + d_3 + d_4 + d_5) + (d_1 + d_2 + d_3 + d_5) + (d_1 + d_2 + d_3 + d_4) + (d_3)$$
$$= (d_1 + d_2 + d_4 + d_5) + (d_1 + d_2 + d_3 + d_5) + (d_1 + d_2 + d_3 + d_4) + (d_2)$$

In the aforementioned strategies A and B, $k-1$ nodes are involved during the course of reconstruction of the first node, i.e., the parity node for strategy A and the systematic node for strategy B. Then, this node together with two other nodes is being used to reconstruct the neighboring node. Therefore, a repair bandwidth of size $\frac{(k-1)M}{k} + \frac{3M}{k} = \frac{(k+2)M}{k}$ is consumed to repair two failed nodes from the same partition. In fact, the average repair bandwidth for reconstructing each node is $\frac{(k+2)M}{2k}$.

As an example, referring to Fig. (2), in the proposed non-MDS$(10,5)$ code, when two nodes containing packets d_1 and $d_2 + d_3 + d_4 + d_5$ simultaneously fails, 7 nodes are totally involved for reconstructing the failed nodes, meaning 3.5 nodes per packet are involved, thereby consuming a repair bandwidth of size $\frac{7M}{5}$ ($\frac{3.5M}{5}$ for each packet). Recall that in a MSR(10,5) code, a new node is allowed to connect to at least five nodes which leads to a repair bandwidth of size M per each failed node.

As is mentioned earlier, there are two possible choices to reconstruct a failed node. First, the new node can connect to $k-1$ nodes from other partitions, or it can connect to its neighboring node together with two nodes from other partitions. Thus, for two specific cases of $k = 2$ and $k = 3$, the reconstruction of a lost packet through connecting to $(k-1)$ nodes is more efficient than the second method, i.e., connecting to three nodes including the neighboring node, since in this case $\frac{(k-1)M}{k}$ is smaller than $\frac{3M}{k}$.

Finally, it should be noted in the proposed non-MDS(2k,k) code, the total storage size needed to store a file of size M is $2M$ which is independent of k. On the other hand, the repair bandwidth is shown to decrease as k increases. As a result, for a given total storage capacity, there is a tradeoff between the repair bandwidth and the number of storage nodes, i.e., $n = 2k$. Moreover, the number of nodes which are involved during the course of repairing a single failure is shown to be at most three.

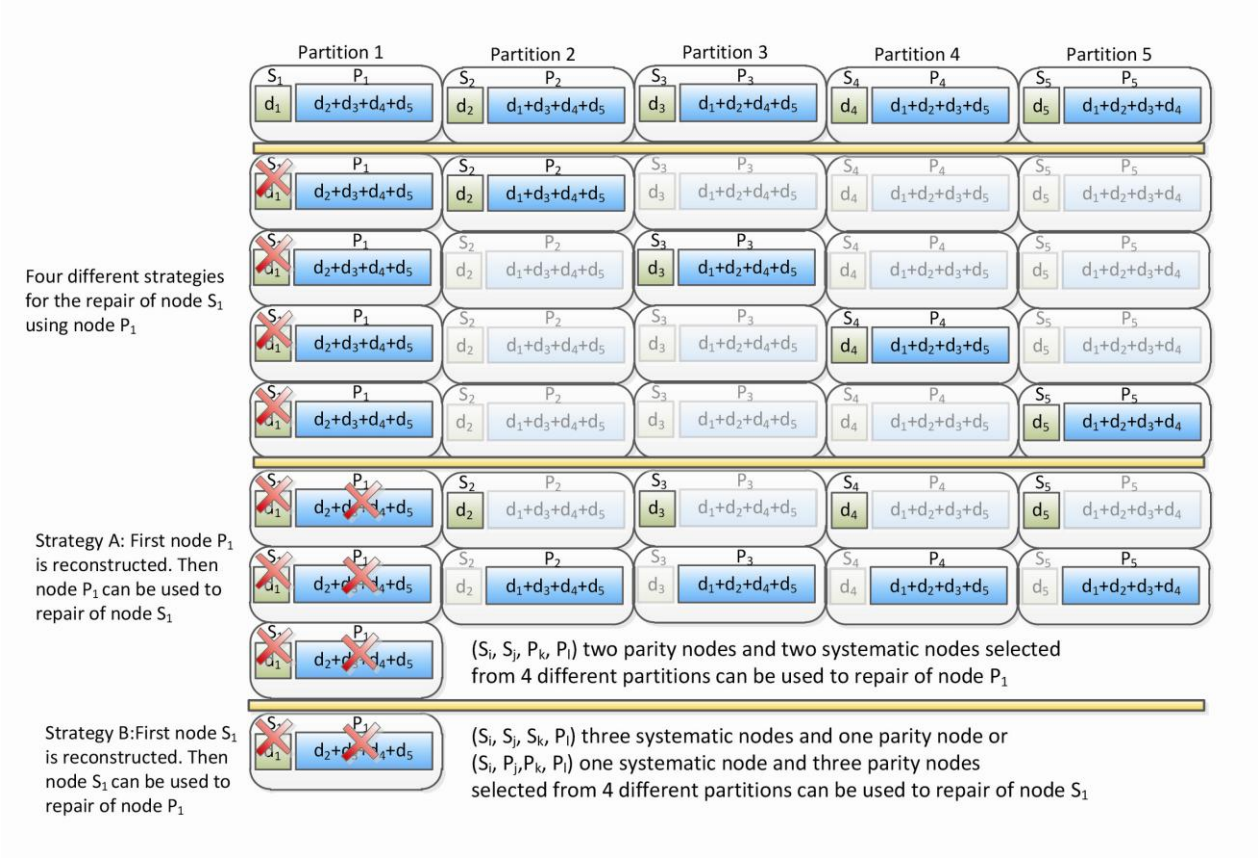

Fig. 2. The repair problem of a $(n, k) = (10,5)$ code. The lost packet d_1 can be repaired by the use of three packets including its related parity packet i.e, $d_2 + d_3 + d_4 + d_5$. Also when $d_2 + d_3 + d_4 + d_5$ has failed d_1 can be reconstructed by the use of four nodes from another partitions.

IV. CONCLUSION

A novel non-MDS code for storage systems is proposed which is shown is able to simultaneously tolerate any three node failures. Also, it can tolerate any $k - 1$ node failures if no more than two failed nodes are from the same partition. Moreover, each single node failure can be repaired through connecting to just three nodes. On the other hand, it is shown the proposed code achieves lower repair bandwidth as compared to MSR codes when there is a restriction on the number of surviving nodes. Finally, the suggested code has a simple structure as each node merely stores one packet and the recovery of the original data file as well as reconstruction a lost packet can be achieved by applying a simple XOR operation on the stored packets.

REFERENCES

[1] M. Blaum and Saumya R.M. Roth, "On lowest density mds codes," *IEEE Transactions on Information Theory*, vol. 45, no. 1, pp. 45–59, January 1999.

[2] D.Patterson, G.Gibson, and R.Katz, "A case for redundant arrays of inexpensive disks (raid)," in *Proc. ACM SIGMODInternational Conference on Management of Data*, Nagoya, Japan, 1988, pp. 109–116.

[3] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and AppliedMathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[4] M. Blaum, J. Brady, J. Bruck, and J. Menon, "Evenodd: An efficient scheme for tolerating double disk failures in raid architectures," *IEEE Transactions on Computers*, vol. 44, no. 2, pp. 192–202, February 1995.

[5] J. L. Hafner, "Weaver codes: Highly fault tolerant erasure codes for storage systems," in *Proc. the 4rd USENIX Symposium on File and Storage Technologies (FAST 2005)*, 2005, pp. 212–224.

[6] Kevin M. Greenan, Xiaozhou Li, and Jay J. Wylie, "Flat xor-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs," *in Proc. the 26th IEEE Symposium on Massive Storage Systems and Technologies (MSST2010)*, Nevada, USA, May 2010.

[7] A. G. Dimakis, P. G. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, September 2010.

[8] K.V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Exact regenerating codes for distributed storage," in *Proc. Allerton Conference on Control, Computing, and Communication*, Urbana-Champaign, IL, September 2009.

[9] A. Kiani and S. Akhlaghi, "Selective regenerating codes," *IEEE Communications Letters*, vol. 15, no. 8, pp. 854–856, August 2011.

[10] Soroush Akhlaghi, Abbas Kiani, and Mohammad Reza Ghanavati, "Cost-bandwidth tradeoff in distributed storage systems," *Computer Communications*, vol. 33, no. 17, pp. 2105–2115, 2010.