LETTER

Single Failure Recovery Method for Erasure Coded Storage System with Heterogeneous Devices

Yingxun FU[†a)], **Junyi GUO**[†], **Li MA**[†], *Nonmembers*, *and* **Jianyong DUAN**[†], *Member*

SUMMARY As the demand of data reliability becomes more and more larger, most of today's storage systems adopt erasure codes to assure the data could be reconstructed when suffering from physical device failures. In order to fast recover the lost data from a single failure, recovery optimization methods have attracted a lot of attention in recent years. However, most of the existing optimization methods focus on homogeneous devices, ignoring the fact that the storage devices are usually heterogeneous. In this paper, we propose a new recovery optimization method named HSR (Heterogeneous Storage Recovery) method, which uses both loads and speed rate among physical devices as the optimization target, in order to further improve the recovery performance for heterogeneous devices. The experiment results show that, compared to existing popular recovery optimization methods, HSR method gains much higher recovery speed over heterogeneous storage devices.

key words: *storage system, erasure code, heterogeneous devices, single failure recovery*

1. Introduction

With the data rapid growth in both size and complexity, data centers have more and more storage devices [1]. Since storage devices usually have a certain error rate, when the scale of devices is very large, the number and possibility of error emergence are very high [2]. In order to recover the lost data from failed devices, today's data center usually adopts erasure codes, which usually divide data into a series of chunks with a fixed size. Erasure codes also provide some redundant chunks calculating from some of the data chunks [3], and then store them into a certain storage system named erasure coded storage system. When some data chunk lost, erasure coded storage system should retrieve a part of surviving data/redundant chunks to reconstruct the lost data.

When suffers from devices errors, the system should fast recover the lost chunks and rewrite them into a new device, in order to maintain the data reliability. The recovery performance is very important for storage systems, which attracts a lot of attention in recent years. Since more than 99.75% failure patterns are single device failure [4], most of the existing researches focus on single failure. Due to the different feature of scenarios and applications, the system adopts different erasure codes. Existing single failure optimization methods usually fit for most of the popular erasure codes, including [5]'s method, BP method [6], STP method [7], et al. Although all these methods usually perform well on homogeneous devices, most of existing methods may not provide good performance when comes to heterogeneous devices, because the bottleneck turns into the slowest device.

Focusing on this problem, in this paper we propose a new heterogeneous device based recovery method termed HSR (Heterogeneous Storage Recovery) method, in order to further improve the recovery performance for heterogeneous devices. HSR method first tests the speed rate of all devices as an input parameter, and then uses simulated annealing algorithm to balance the production of the loads and speed rate among all devices. The experiment results show that, compared to [5]'s method and STP method, HSR method gains up to 39.6% and 17.6% higher recovery speed over existing popular erasure codes with heterogeneous devices, respectively.

2. Background and Our Motivation

2.1 Terms and Notations

We first give some frequently used terms and notations based on [8]. In erasure coded storage systems, data and parity that calculate by a set of data to improve the reliability, are usually partitioned to chunks and stored in different physical devices. These chunks termed "elements", including data elements and parity elements. The basic logic unit of erasure coded system names "stripe". In each stripe, the generation of the parity elements only depends on the elements of this stripe. Each column of "stripe" calls "strip". To assure the reliability, each strip of the same stripe should be mapped to different physical devices. We use $d_{i,j}$ and $p_{i,j}$ to denote the ith element of the jth column, where $d_{i,j}$ represents data element, and $p_{i,j}$ denotes the parity element. We use D_i and P_i to represent data strip and parity strip, respectively. Figure 1 shows an example of one stripe.

2.2 Erasure Codes

Erasure codes are widely used in reliable storage systems, including Reed-Solomon code [9], Rotated Reed-Solomon code [5], et al. These codes are $GF(2^n)$ based codes usually tolerating arbitrary device failures. The other important class of erasure codes is XOR-based codes, including Cauchy Reed-Solomon code [10], Liberation code [11], RDP code [16], D-Code [13], et al. Cauchy Reed-Solomon

Manuscript received January 24, 2019.
Manuscript revised March 29, 2019.
Manuscript publicized June 14, 2019.

[†]The authors are with the College of Computer Science and Technology, North China University of Technology, China.
a) E-mail: fuyx@ncut.edu.cn

DOI: 10.1587/transinf.2019EDL8021

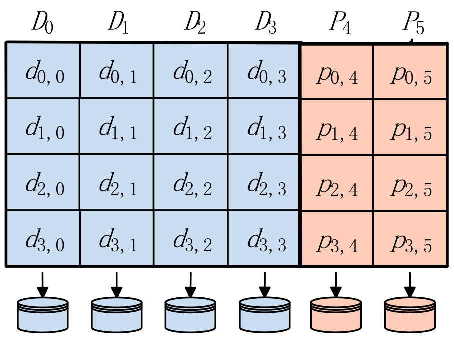

D_0 D_1 D_2 D_3 P_4 P_5
$d_{0,0}$ $d_{0,1}$ $d_{0,2}$ $d_{0,3}$ $p_{0,4}$ $p_{0,5}$
$d_{1,0}$ $d_{1,1}$ $d_{1,2}$ $d_{1,3}$ $p_{1,4}$ $p_{1,5}$
$d_{2,0}$ $d_{2,1}$ $d_{2,2}$ $d_{2,3}$ $p_{2,4}$ $p_{2,5}$
$d_{3,0}$ $d_{3,1}$ $d_{3,2}$ $d_{3,3}$ $p_{3,4}$ $p_{3,5}$

Fig. 1 An example of one stripe.

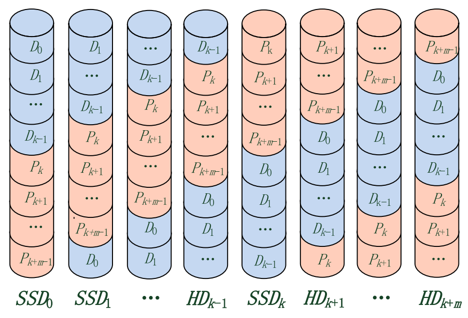

SSD_0 SSD_1 ··· HD_{k-1} SSD_k HD_{k+1} ··· HD_{k+m}

Fig. 3 An example of heterogeneous erasure coded system.

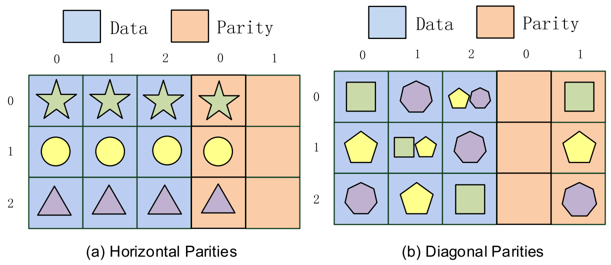


Fig. 2 An example of 5-devices liberation code.

code is an improved Reed-Solomon code tolerating arbitrary number of device failures. Liberation code, RDP code, and D-Code are classic RAID-6 codes tolerating up to 2 device failures.

Figure 2 gives an example of 5-devices Liberation code, where the same icons mean the XOR sum of the related elements equals to zero. E.g., in Fig. 2 (a), $d_{0,0}$, $d_{0,1}$, $d_{0,2}$, and $p_{0,0}$ with the same icon "star", which means $d_{0,0} \oplus d_{0,1} \oplus d_{0,2} \oplus p_{0,0} = 0$ or $p_{0,0} = d_{0,0} \oplus d_{0,1} \oplus d_{0,2}$. Similarly, in Fig. 2 (b), $d_{0,0} \oplus d_{1,1} \oplus d_{2,2} \oplus p_{0,1} = 0$. Based on the encoding equations, we can choose either one of the above equations to recover $d_{0,0}$ when D_0 fails, which is the basic principle of single failure recovery optimization.

2.3 Heterogeneous Erasure Coded Storage Systems

Today's storage systems usually suffer from storage device failures. When a device fails, the system should recover the data from the survived devices and use new devices to replace the failed device to establish the new erasure coded system. The new device and failed device have different types and specifications forming the heterogeneous storage systems. On the other hand, erasure coded storage system usually rotate the map from strip to physical devices to alleviate the influence of writes. Figure 3 gives an example of heterogeneous erasure coded storage systems, where the access speed rate for solid state disks (SSDs) are higher than hard disks (HDs).

2.4 Single Failure Recovery Methods

When suffer from device failures, erasure coded storage systems should retrieve the surviving information to recover the lost elements. Since more than 99.75% failure patterns are a single failure, recently researches usually focus on single failure recovery, in which the optimization methods including [5]'s recovery method, BP method [6], STP method [7], SIOR method [14], SSDM method [12], et al. These methods use hybrid recovery principle [5], in order to search for the proper recovery scheme over different scenarios.

2.5 Our Motivation

Existing single failure recovery methods usually focus on homogeneous devices and assume that the storage devices have similar access speed [7], [12], [14], so that the bottleneck is the highest loaded device. In heterogeneous storage systems, the lowest accessed device is usually not the most loaded device but the oldest device. We should propose new recovery optimization method for heterogeneous storage systems, in order to balance the access speed and loads among all devices.

3. The HSR Recovery Method

In this paper, we propose a new recovery method named heterogeneous storage recovery method (HSR method), in order to speed up the recovery performance on erasure coded storage system over heterogeneous devices. HSR method first tests the access speed for each device as an input parameter, and then utilizes the recovery equations and simulation annealing algorithm to search for the proper recovery scheme.

3.1 Generate Recovery Equations

We first generate recovery equations for optimizing the recovery performance. Firstly, we generate the original equations by the encoding rules. We then generate the derived equations by iterating the common elements from different equations. Repeat this process, until no more new equations have been generated. Algorithm 1 gives the pseudo-code.

Line 1 defines two parameters E_{all} and E_{useful} to deposit the recovery equations. Line 2 generates the original recovery equations by the encoding rules of the erasure code. Line 3–14 define a while loop, which is used for

Algorithm 1 Generate Recovery Equations

```
1: initial $E_{all}$, $E_{useful}$
2: $E_{all} \leftarrow$ encoding rules
3: while no new equation added to $E_{all}$ do
4:     for Each $equ_i \in E_{all}$ do
5:         for Each $ele_i \in e_i$ do
6:             for Each $equ_j \in E_{all}$ do
7:                 if $ele_i \in equ_j$ then
8:                     $new_{equ} = equ_i$ - $ele_i$ + $equ_j$ - $ele_i$
9:                     $E_{all}$.add($new_{equ}$)
10:                end if
11:            end for
12:        end for
13:    end for
14: end while
15: for Each $equ_i \in E_{all}$ do
16:     for Each $ele_i$ in the lost element set do
17:         if $ele_i \in equ_i$ then
18:             $E_{useful}$.add($equ_i$)
19:         end if
20:     end for
21: end for
22: return $E_{useful}$
```

finding the recovery equations and storing them into E_{all}. Specifically, Line 4–5 sequently go through all elements of each equation in E_{all}. Line 6 go through all equations in E_{all}. Line 7–8 indicate that if equ_i and equ_j simultaneously contain ele_i, then combine the two equations into a new equation by replacing ele_i. Line 9 adds the new equation into E_{all}. When E_{all} does not add any new equation for the loop, then end the loop. Line 15–21 utilize dual-for loop to pick out the equations containing lost element, and then store them in E_{useful}. The algorithm returns E_{useful}.

3.2 Search for Recovery Scheme

We then utilize the E_{useful} to search for the proper equations (i.e., recovery solutions) to reconstruct the lost elements. The basic principle is to set an array of parameters “weights” to control the “energy”, and use the standard simulated annealing algorithm to search for the feasible recovery solutions. Algorithm 2 gives the pseudo-code.

Line 1 collects the speed rate among all storage devices. Line 2 initials two key parameters for the algorithm. Line 3 defines a function, which uses to calculate the energy for simulated annealing. Line 4–7 count the number of the required elements of each $strip_i$, and then multiples r_i as the weight. Line 8 returns the square deviation of weight as energy.

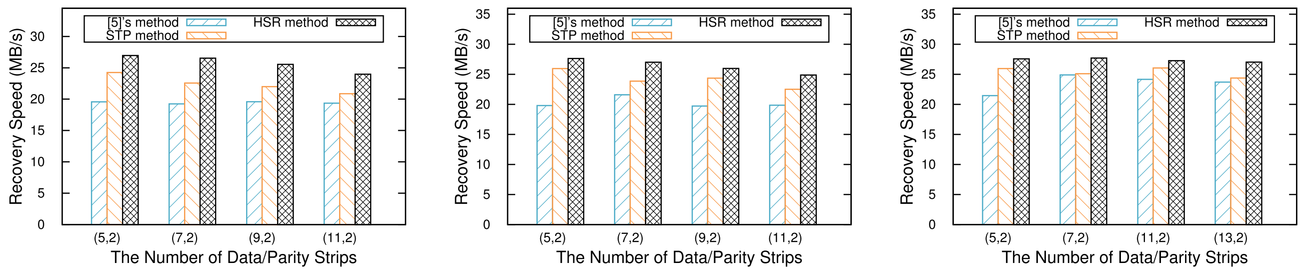
Line 10–31 use simulated annealing algorithm to search for the recovery scheme. We first initiate three parameters (K, T, M) for standard simulated annealing algorithm, and initiate parameter R to control the variation from *Solution* to $Solution_{new}$ (Line 10). Line 11 generates the initial *Solution* from E_{useful}. Line 12–13 calculate the energy of *Solution* and set the expire condition. The following steps are used to optimize the *Solution*. First, we randomly select R failed elements and denote their relative recovery equations in *Solution* as $E_{need\_replace}$, and then generate a replacement equation set E_{new} by choosing another recovery equation for each selected failed elements. We replace each equation of $E_{need\_replace}$ by its relative equation in E_{new}, in order to generate another stack-based solution $Solution_{new}$ (Line 15–17). Afterwards, we calculate the energy gap between $Solution_{new}$ and *Solution*, and utilize simulated annealing algorithm to determine whether replace *Solution* with $Solution_{new}$ or not (Line 18–19). Line 20–30 use standard simulated annealing algorithm to optimize the *Solution*. Specifically, Line 20–23 illustrate if the energy subtraction is above zero or a random condition has been triggered, the algorithm will replace the *Solution* by $Solution_{new}$, and reset *remain_times* to M. If the energy subtraction is below zero and the replacing process occurs, the algorithm reduces the temperature by parameter K and T (Line 24–26). Else if the replacing process does not occur, the algorithm decreases the *remain_times*, until the *remain_times* equals to zero (Line 27–29). At last, the algorithm returns *Solution* (Line 31).

Algorithm 2 Search for Recovery Solution

```
1: use $r_i$ to represent the access rate for each device
2: initial parameter $weight$, $strip$
3: procedure CAL_ENERGY($Solution$)
4:     for each $strip_i \in Solution$ do
5:         $num_i \leftarrow$ the number of need element in $strip_i$
6:         $weight_i = num_i \times r_i$
7:     end for
8:     return the square deviation of $weight$
9: end procedure
10: initial parameter $E_{useful}$, $K$, $T$, $R$ and $M$
11: $Solution \leftarrow E_{useful}$
12: $de = cal\_energy(Solution)$
13: $remain\_times = M$
14: while $remain\_times > 0$ do
15:     $E_{need\_replace} \leftarrow$ Select $R$ equations in $Solution$
16:     $E_{new} \leftarrow select\_equations(E_{need\_replace}, E_{useful})$
17:     $Solution_{new} = generate\_solutions(Solution, E_{new})$
18:     $de_{new} = cal\_energy(Solution_{new})$
19:     $\Delta t = de_{new} - de$
20:     if $\Delta t > 0$ or $exp(\Delta t/T) > rand()/rand\_max$ then
21:         $Solution = Solution_{new}$
22:         $de = de_{new}$
23:         $remain\_times = M$
24:         if $\Delta t < 0$ then
25:             $T = K * T$
26:         end if
27:     else
28:         $remain\_times - -$
29:     end if
30: end while
31: Return $Solution$
```

3.3 Reconstruct the Lost Elements

According to the *Solution* returned in Algorithm 2, we can first retrieve the needed elements from surviving devices in parallel. We then reconstruct the lost elements by the related equations in *Solution*, and write the reconstructed elements into new storage devices.



(a) Reed-Solomon Code.

(b) Cauchy Reed-Solomon Code.

(c) Liberation Code.

Fig. 4 Recovery speed for different erasure codes over heterogeneous storage devices.

Table 1 Time overhead for different recovery methods over different erasure codes (unit: seconds).

Erasure Codes	Ref. [5]	STP	HSR
RS $(9,2)$	< 1s	< 1s	< 1s
RS $(10,3)$	< 1s	< 1s	< 1s
Liberation $(11,2)$	< 1s	2s	2s
Liberation $(13,2)$	< 1s	3s	3s
Cauchy RS $(9,2)$	< 1s	3s	3s
Cauchy RS $(10,3)$	< 1s	8s	8s
Cauchy RS $(13,3)$	1s	24s	23s
Cauchy RS $(16,4)$	941s	232s	229s

4. The Search Time Analysis

We analyze the search time of HSR method by comparing with Ref. [5]'s method and STP method [7], which are popular recovery method for today's erasure coded storage systems. Reference [5]'s method searches for all potential stripe-based solutions for each stripe, so that the search time is exponentially increased with the scale. STP method and HSR method utilize simulated annealing algorithm to search for the feasible solutions in the stack-level. The time cost is major in the initial process of simulated annealing algorithm, which is polynomially increased with the scale. Therefore, when the scale is small, Ref. [5]'s method will run fast due to the faster initial process; otherwise, STP method and HSR method run fast due to the polynomial complexity. STP method and HSR method have similar time overhead, because they have the similar scale and utilize the same algorithm concept.

We implement Ref. [5]'s method, STP method, and our proposed HSR method based on C++ language over visual studio 2010 toolbox, and use RS code, and Cauchy RS code [10], and Liberation code with different parameter to evaluate the real search time. The machine has a Intel i7-6820HQ CPU and 32GB memory. As shown in Table 1, the search time well matches above analysis and illustrates that the time complexity is very close to STP method.

On the other hand, the searching process only runs on the initialization of erasure coded storage system with very low frequency, which is not performance sensitive. The recovery process runs when the system suffers from disk failure, which is important for erasure coded storage systems as discussed in Sect. 1. Therefore, we usually pay more attention on recovery performance.

5. Experiment Evaluation

In this section, we conduct a series of experiments in real storage system, in order to illustrate HSR method achieves good single failure recovery performance over heterogeneous devices.

5.1 Environment

Our experiments are run on a machine and a disk array. The hardware environment of the machine is an Intel Xeon X5472 processor and 12GB memory. The disk array contains 13 Seagate 10K.3 SAS disks and 3 SATA-3 Disks. Each SARS disk has 300GB capability, and each SATA3 disk has 1TB capability. The machine and disk array are connected by a fiber cable with 800MB bandwidth. The operation system of the machine is SUSE with Linux fs91 3.2.16.

5.2 Methodology

We select Reed-Solomon coded, Cauchy Reed-Solomon coded, and Liberation coded storage systems as comparison, and implement them with different parameter in the above environment by Jeasure-1.2 [15], which is widely used for erasure coded storage system implementation. On the other hand, we implement [5]'s method, STP method [7] and our proposed HSR method by C++ language over visual studio 2010 toolbox in the environment referred in Sect. 4. Please note that the process of searching recovery solutions runs when the erasure coded storage system initializes. The recovery solutions are usually stored in both memories and permanent storage devices.

When suffer from disk failures, the storage system should retrieve the recovery solutions from memories or permanent storage devices, and access the surviving elements to reconstruct the lost elements. We test all potential physical failures for all test erasure coded storage systems, collect the real recovery speed, and calculate the average recovery speed as evaluation metrics. For the tested erasure coded storage systems, we set 3 SATA disks and some SARS disks as heterogeneous devices, and use the circularly rotated mappings referred in Fig. 3. We set Reed-Solomon code and Cauchy Reed-Solomon code with two parity strips.

In addition, we consider 20 stacks for each tested cases, so that we have to recover up to 87.04GB data from the failed physical disk.

5.3 Result Analysis

Figure 4 shows the evaluation results for the tested erasure codes. We can easily observe that HSR method provides higher recovery speed than Ref. [5]'s method and STP method. This is because HSR method balance the number of elements and speed rate among physical storage devices simultaneously, but other methods only consider homogeneous devices in which the storage devices have the similar speed rate. In statistic, HSR method gains 11.3% to 39.6% higher recovery speed than Ref. [5]'s method, and achieves 7.6% to 14.7% higher recovery speed than STP method for the tested erasure codes over heterogeneous devices. The results illustrate that HSR method has good recovery performance than other tested recovery methods.

6. Conclusion

In this paper, we propose a new recovery optimization method termed HSR (heterogeneous storage recovery) method, in order to optimize the single failure recovery performance for erasure coded storage systems with heterogeneous devices. HSR method uses the production of the loaded and the speed rate among all disks as the optimization target, and utilizes the concept of simulated annealing algorithm to search for the proper recovery scheme. The experiment results show that HSR method gains much higher recovery speed compared to existing popular recovery optimization methods over heterogeneous storage devices.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61702013, 61672040), the Beijing Outstanding Talent Training Project (2016000020124G016), the Science and Technology Project of Beijing Educational Committee (KM201710009008, KZ201810009011), and the Science and Technology Innovation Project of North China University of Technology (18XN012, 18XN053).

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," Proc. ACM SOSP '03, pp.29–43, 2003.

[2] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. Fahim ul Haq, M. Ikram ul Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas, "Windows azure storage: A highly available cloud storage service with strong consistency," Proc. ACM SOSP '11, pp.143–157, 2011.

[3] M. Blaum, J. Brady, J. Bruck, and J. Nebib, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," IEEE Trans. Inf. Theory, vol.44, no.2, pp.192–202, 1995.

[4] E. Pinheiro, W. Weber, and L. Barroso, "Failure trends in a large disk drive population," Proc. USENIX FAST '07, 2007.

[5] O. Khan, R. Burns, J. Plank, and W. Pierce, "Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads," Proc. USENIX FAST '12, 2012.

[6] Y. Fu, J. Shu, and X. Luo, "A stack-based single disk failure recovery scheme for erasure coded storage systems," Proc. IEEE SRDS '14, pp.136–145, Oct. 2014.

[7] Y. Fu, J. Shu, Z. Shen, and G. Zhang, "Reconsidering single disk failure recovery for erasure coded storage systems: Optimizing load balancing in stack-level," IEEE Trans. Parallel Distrib. Syst., vol.27, no.5, pp.1457–1469, 2016.

[8] J.L. Hafner, V. Deenadhayalan, T. Kanungo, and K. Rao, "Performance metrics for erasure codes in storage systems," Technical Report, RJ 10321, IBM Research, San Jose, 2004.

[9] I.S. Reed and G. Solomon, "Polynomial codes over certain finite fields," J. Society for Industrial and Applied Mathematice, vol.8, no.2, pp.300–304, 1960.

[10] J. Blomer, M. Kalfane, R. Krap, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," Technical Report TR-95-048, International Computer Science Institute, Aug. 1995.

[11] J. Plank, "The RAID-6 liberation codes," Proc. USENIX FAST '08, Feb. 2008.

[12] Y. Fu, S. Wen, L. Ma, and J. Duan, "Strip-switched deployment method to optimize single failure recovery for erasure coded storage systems," IEICE Trans. Inf. & Syst., vol.E101-D, no.11, pp.2818–2822, Nov. 2018.

[13] Y. Fu and J. Shu, "D-Code: An efficient RAID-6 code to optimize I/O loads and read performance," Proc. IPDPS '15, pp.603–612, 2015.

[14] Z. Shen, J. Shu, and Y. Fu, "Seek-efficient I/O optimization in single failure recovery for XOR-coded storage systems," Proc. SRDS '15, pp.228–237, Sept. 2015.

[15] J. Plank, S. Simmerman, and C. Schuman, "Jerasure: A library in C/C++ facilitating erasure coding for storage applications-Version 1.2," Technical Report CS-08-627, University of Tennessee, Aug. 2008.

[16] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," Proc. FAST '04, 2004.