IEICE TRANS. INF. & SYST., VOL.E101–D, NO.11 NOVEMBER 2018

LETTER

Strip-Switched Deployment Method to Optimize Single Failure Recovery for Erasure Coded Storage Systems

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

SUMMARY With the rapid growth on data scale and complexity, single disk failure recovery becomes very important for erasure coded storage systems. In this paper, we propose a new strip-switched deployment method, which utilizes the feature that strips of each stripe of erasure codes could be switched, and uses simulated annealing algorithm to search for the proper strip-deployment on the stack level to balance the read accesses, in order to improve the recovery performance. The analysis and experiments results show that SSDM could effectively improve the single failure recovery performance.

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

1. Introduction

With the rapid growth of data in both scale and complexity, more and more large-scaled systems store their data in massive inexpensive storage devices, such as Google GFS [1] and Windows Azure [2]. However, the inexpensive devices such as disks and SSDs usually suffer from hard failures causing permanently data lost. Storage systems should equip some redundant information, in order to ensure the data could be recovery from storage device failures. Today's storage systems usually adopt erasure codes to maintain the data reliability, which forms the erasure coded storage systems.

When suffer from device failures, erasure coded storage systems should recover the lost data. Current recovery optimizing methods usually focus on optimizing the total data accesses [3], sequentially data accesses [4], and parallel data accesses [5], but they usually work on the stripe level. Though [6], [7] focus on stack level (a series of consecutive stripes) and provide good recovery performance, they only focus on existing stacks and do not construct the rules for generating the stack. [8] provides a construction rules for the generation of stack to improve the recovery performance, but it only fit for X-Coded storage systems.

In this paper, we propose a strip-switched deployment method (SSDM) that fits for most of erasure coded storage systems, in order to optimize the performance on single failure recovery. The proposed SSDM is orthotropic with existing optimizing methods, which first uses one or more pre-defined recovery schemes for each stripe, and then utilizes simulated annealing algorithm to search for the proper strip-deployment on stack level, in order to enhance the parallelism of read accesses, and thus improve the performance. The analysis and experiment results show that, SSDM could effectively improve the recovery performance.

Manuscript received February 8, 2018.
Manuscript revised June 9, 2018.
Manuscript publicized July 25, 2018.

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

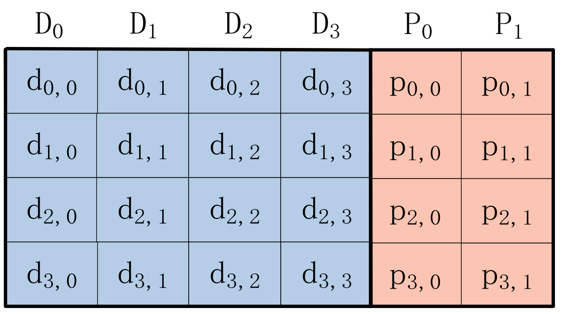
DOI: 10.1587/transinf.2018EDL8031

2. Background and Related Work

Terms and Notations: We first give some frequently used terms and notations based on [7]. The minimal unit of erasure coded storage system terms "stripe", which contains data elements and parity elements, where the "element" is a chunk of information. The generation of the parity elements only depends on the elements of the current stripe. Each column of the "stripe" names "**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 $i-th$ element of the $j-th$ 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.

Erasure Code: Erasure code is a class of error correction code, which is commonly used in storage systems to correct the data from storage device failures. Erasure code is usually represented as one stripe, because the stripes are independent for each other in logic. Figure 2 shows an example of (4,2,2) LRC code.

A stripe of (4,2,2) LRC code has 4 data elements and 4 parity elements, where the first 2 parity elements are calculated based on 2 data elements, respectively, and the other 2 parity elements are computed based on all 4 data elements. E.g., parity element $p_{0,1}$ is calculated based on $d_{0,0}$ and $d_{0,1}$. When suffer from disk failures, the system could reconstruct the lost data elements by their related parity elements and the surviving data elements.



D_0	D_1	D_2	D_3	P_0	P_1
$d_{0,0}$	$d_{0,1}$	$d_{0,2}$	$d_{0,3}$	$p_{0,0}$	$p_{0,1}$
$d_{1,0}$	$d_{1,1}$	$d_{1,2}$	$d_{1,3}$	$p_{1,0}$	$p_{1,1}$
$d_{2,0}$	$d_{2,1}$	$d_{2,2}$	$d_{2,3}$	$p_{2,0}$	$p_{2,1}$
$d_{3,0}$	$d_{3,1}$	$d_{3,2}$	$d_{3,3}$	$p_{3,0}$	$p_{3,1}$

Fig. 1 An example of one stripe.

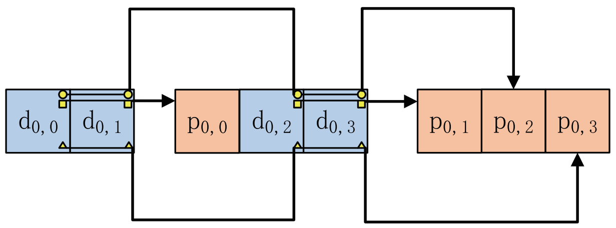


Fig. 2 An example of (4,2,2) LRC code.

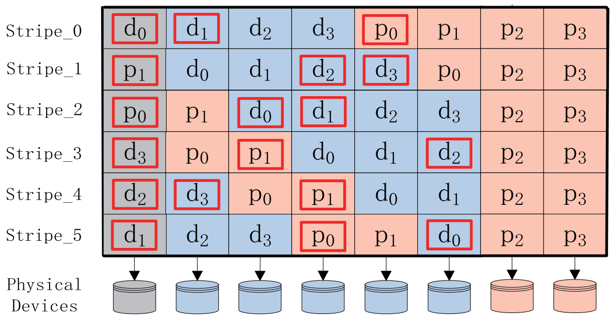


Fig. 3 An example of rotated mapping and single failure recovery for (4,2,2) LRC code.

Erasure Coded Storage System: The storage system adopts erasure code techniques names erasure coded storage system. The construction of erasure coded storage system is to map the stripes into physical storage devices. In order to assure the data reliability, each strip of the same stripe should be mapped to different devices. The system assigns a fixed size (i.e., 1MB) for each element, and then stores the data and parities stripe by stripe. Strips could be mapped to different devices.

Figure 3 shows an example of rotated mapping for (4,2,2) LRC code. We use the lowercase and strip number to represent both strip and element, because each strip of LRC code has only one element. For example, the d_0 and p_0 in Fig. 3 is equivalent to $d_{0,0}$, $p_{0,0}$ in Fig. 2 and D_0, P_0 in Fig. 1, respectively.

Single Failure Recovery: When suffer from disk failures, erasure coded storage systems should retrieve the surviving information to recover the lost data/parity elements. Since most of device failure patterns are single failure (more than 99.75%) [7], [9], researchers usually focus on single failure recovery, including recovery optimizing methods [3], [4] and new erasure codes [3], [10] that aim to improve the recovery performance. Figure 3 also shows a recovery example for (4,2,2) LRC code [10] when the first device fails. The system uses p_0 and p_1 to recover the lost data elements, and uses the constructing rules reconstructing parity elements, until all lost elements are reconstructed.

3. Design of SSDM

In this paper, we propose a new recovery optimizing method named stripe-switched deployment method (SSDM), in order to further optimize the performance on single failure recoveries. SSDM utilizes the feature that strips of each stripe of erasure codes could be switched, and then adopts simulated annealing algorithm to search for the proper strip deployment on the stack level to balance the read accesses.

Algorithm 1 Strip Switched Deployment Method

```
1: initial stripe recovery methods $R_i$
2: initial parameters $K, T, M$
3: for $stripe_{index} = 0$ to $2 \cdot strip\_number$ do
4:     generate $Stack$ contains $stripe_{index}$ stripes
5:     $Solution \leftarrow R_i$
6:     $de = cal\_energy(Solution, Stack)$
7:     $remain\_times = M$
8:     while $remain\_times > 0$ do
9:         generate random numbers $a, b, c$
10:        $Stack_{new} = stack\_switch(a, b, c)$
11:        $Solution_{new} = solution\_switch(a, b, c)$
12:        $de_{new} = cal\_energy(Solution_{new}, Stack_{new})$
13:        $\Delta t = de_{new} - de$
14:        if $\Delta t > 0$ or $exp(\Delta t/T) > rand()/rand\_max$ then
15:            $Stack = Stack_{new}$
16:            $Solution = Solution_{new}$
17:            $de = de_{new}$
18:            $remain\_times = M$
19:            if $\Delta t < 0$ then
20:                $T = K * T$
21:            end if
22:        else
23:            $remain\_times - -$
24:        end if
25:    end while
26:    $Stack_{all}.add(Stack)$
27:    $Solution_{all}.add(Solution)$
28: end for
29: Return $Stack_{all}$ and $Solution_{all}$
```

3.1 Pre-Define Recovery Scheme for Each Stripe

SSDM adopts pre-defined recovery schemes for each stripe, which implies SSDM is orthogonal with most of existing recovery optimizing method. For example, we could choose local parities for LRC code and the schemes optimized by the methods in [3], [5] for XOR-based erasure codes, acting as SSDM's pre-define recovery schemes.

3.2 Generate the Switched Strip-Deployment

SSDM uses the simulated annealing algorithm to produce stacks. Simulated annealing algorithm is a generic probabilistic algorithm for addressing the global optimization problem in a large and discrete solution space, which is usually more efficient than enumeration-based algorithms, because it only search for the acceptable solution. Algorithm 1 gives the details.

In SSDM algorithm, line 1 initials the stripe recovery methods for all strip broken cases. E.g., if a stripe contains n strips, then R_i has n different recovery methods representing the potential n possible strip broken cases, where $0 \leqslant i \leqslant n-1$. Line 2 initials the key parameters of simulated annealing algorithm, where K and T control the temperature decreasing, and M means the end condition. Line 3 launches a for cycle, which tries that one stack contains 1 to $2 \cdot n$ stripes presenting by $stripe_{index}$. Line 4-5 generate the initial stack ($Stack$) and the initial recovery solution for the stack ($Solution$). Line 6 calculates the energy of the initial stack and recovery solution based on the ”balance” among disks. Line 7 sets the variable $remain\_times$ to record the remaining iteration times. Line 9-11 generate three random numbers a, b, c, which means to switch the $b-th$ strip and $c-th$ strip in the $a-th$ stripe, and then adjust these strips in the stack-based recovery solution. Line 12-13 calculate the new energy and the energy subtraction. Line 14-25 give the simulated annealing process. Specifically, line 14-18 illustrate that if the energy substraction is above zero or a random condition has been triggered, the algorithm will replace the $Stack$ and $Solution$ by $Stack_{new}$ and $Solution_{new}$, respectively, and reset $remain\_times$ to M. If the energy subtraction is below zero and the replacing process occurs, SSDM then reduces the temperature by parameter K and T (line 19-21). Else if the replacing process does not occur, SSDM then decreases the $remain\_times$, until the $remain\_times$ equals to zero (line 22-24). At last, the algorithm produces $Stack$ and $Solution$ for the stack contains from 1 to $2 \cdot n$ stripes, SSDM adds them into $Stack_{all}$ and $Solution_{all}$ and returns $Stack_{all}$ and $Solution_{all}$ (line 26-29). System designers could choose proper $Stack$ and $Solution$ to design the system to further improve the performance on single disk failure recoveries.

3.3 An Example of (4,2,2) LRC Code

We now give a (4,2,2) LRC code example. Figure 4 gives the deployment of one stack produced by SSDM. Since the last two physical disks for LRC code store the global parities, which are used for recovery from more than one disk failures, we store them in special parity disks that may locate far away from the data. SSDM algorithm generates a stack with 10 stripes, where the strip-deployment of each stripe are different. We do not consider about the load balancing on write operations, because LRC codes are usually used for cloud storage systems in which the random write operations are not permitted.

When suffers from a single disk failure, if the failed disk is global parity disk, the system only need to find a leisure time to recompute the parities, because this failure pattern is not important for erasure coded storage systems due to the fact that all data elements are surviving. Otherwise, the system needs to reconstruct the lost elements by local parities. Figure 4 also shows the recovery process from the first disk failure. We can easily observe that all disks except for the failed disk and global parity disks should read 4 elements, which is well balanced. In fact, no matter which data disk fails, all other data disks need to read only 4 elements for recovery under this pattern.

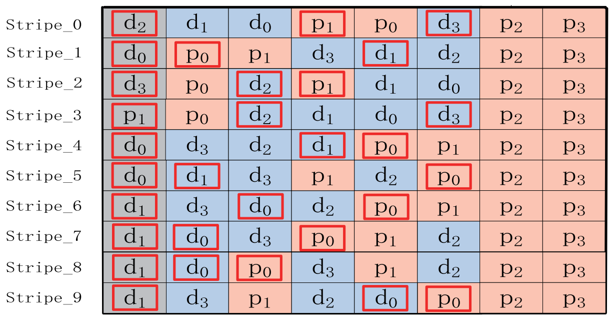

Stripe_0 d2 d1 d0 p1 p0 d3 p2 p3
Stripe_1 d0 p0 p1 d3 d1 d2 p2 p3
Stripe_2 d3 p0 d2 p1 d1 d0 p2 p3
Stripe_3 p1 p0 d2 d1 d0 d3 p2 p3
Stripe_4 d0 d3 d2 d1 p0 p1 p2 p3
Stripe_5 d0 d1 d3 p1 d2 p0 p2 p3
Stripe_6 d1 d3 d0 d2 p0 p1 p2 p3
Stripe_7 d1 d0 d3 p0 p1 d2 p2 p3
Stripe_8 d1 d0 p0 d3 p1 d2 p2 p3
Stripe_9 d1 d3 p1 d2 d0 p0 p2 p3

Fig. 4 Deployment and recovery from the first failure for (4,2,2) LRC code over SSDM algorithm.

4. Simulation Analysis

4.1 Analysis of Search Time

We analyze the time complexity of SSDM by comparing with reference [3]'s method and STP method [7], which are popular recovery method for today's erasure coded storage systems. Reference [3]'s method is stripe-based, which searches for all potential stripe-based solutions for each stripe, so that the search time will be exponentially increased with the scale. STP method and SSDM utilize simulated annealing concept to search for feasible solution in stack-level, where the time cost is major in the initial process of simulated annealing and usually polynomially increased with the scale. If the scale is small, reference [3]'s method runs fast; otherwise, STP method and SSDM run fast. STP method and SSDM have similar time cost, because they have similar scale and utilize the same concept.

We also implement reference [3]'s method, STP method, and our proposed SSDM based on C++ language over visual studio 2010, and use LRC codes and Cauchy RS codes [11] with different parameter to evaluate the real search time. The machine has a Intel i7 6820 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, since the search algorithm only runs on the stage of storage system initialization with very low frequency, the search time of SSDM is acceptable for today's storage systems.

4.2 Analysis of Recovery Speed

We compare the single disk recovery balance among LRC codes and Cauchy RS codes with standard form, rotated form (as shown in Fig. 3) and SSDM based form, and implement them by Jerasure library [12], which is commonly used for erasure coded storage system implementations. We use notation u and v to represent the number of stripes in

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

Erasure Codes	Ref. [3]	STP	SSDM
LRC (8,2,3)	< 1s	< 1s	1s
LRC (10,2,4)	< 1s	< 1s	2s
Cauchy RS (11,2)	< 1s	8s	10s
Cauchy RS (13,3)	1s	26s	31s
Cauchy RS (16,4)	943s	234s	247s

one stack and the number of disks storing the data and local parities, respectively, and define max_read_i as the maximum number of reads among all disks when the $i-th$ ($0 \leqslant i \leqslant v$) disk fails. We define $recovery\_balance$ based on Eq. (1).

$$recovery\_balance = \frac{\sum_{i=0}^{v} max\_read_i}{u \cdot v} \tag{1}$$

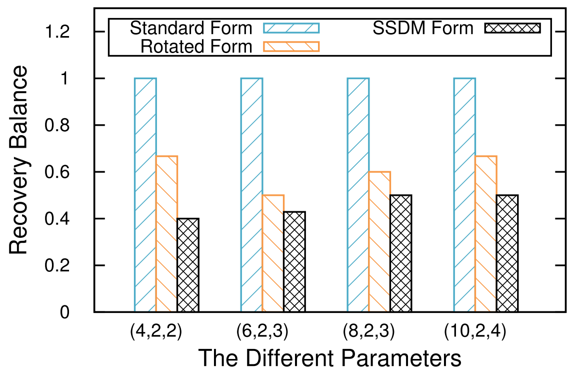
According to Eq. (1), $recovery\_balance$ first calculate the average number of max_read_i, and then divide by u. Since u means the number of failed elements, and v means the number of potential disk failure patterns, $recovery\_balance$ represents the average number of max_read_i for all potential failure cases for recovering each lost element. In today's storage system, all disks could be accessed in parallel, so that the disk with maximum read accesses forms a bottleneck, and the total recovery speed is due to the bottleneck disk accomplishing the read process [7]. Therefore, $recovery\_balance$ could directly reflect the recovery speed.

We first consider LRC codes with parameters (4,2,2), (6,2,3), (8,2,3), and (10,2,4), and show the results in Fig. 5(a). For standard forms, the recovery balance is stabilized in 1, because it only uses the same disks to reconstruct lost data like RAID-4. For rotated forms, the recovery balance is 0.5 to 0.67, which inclines better recovery performance than standard forms, because the more disks participate in recovery process causing better parallelism of read accesses. For SSDM forms, the recovery balance is 0.4 to 0.5, which is 16.7% to 26.7% lower than rotated forms, respectively. In addition, the lower bound of recovery balance is $\frac{\frac{v}{2}-1}{v-1} = \frac{v-2}{2v-2}$, which are 0.4, 0.43, 0.44, 0.45 when v equals to 6, 8, 10, 12. Therefore, For (4,2,2) and (6,2,3) LRC codes, the SSDM forms achieve the lower bounds of the recovery balance. For (8,2,3), and (10,2,4) LRC codes, the recovery balance of SSDM forms are very close to the lower bounds.

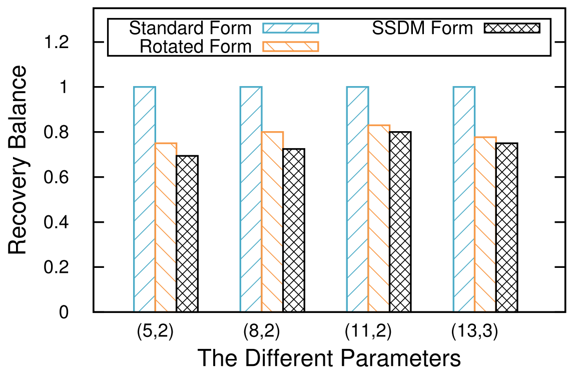
We then consider Cauchy RS codes with different parameters (k, m), where k and m mean the number of data strips and parity strips, respectively. For standard forms and rotated forms, we use reference [3]'s method and STP method [7] for optimization, respectively, and show the result in Fig. 5(b). Compared to rotated forms, SSDM forms achieve 3.8% to 10.4% lower balance, which illustrates that SSDM also performs well on Cauchy RS code.

5. Experiment Evaluation

We build a series of experiments in a real storage system to evaluate the recovery performance. We use the same experiment environment as [6], [7]. The experiments are run on a machine and a disk array. The machine has an Intel Xeon

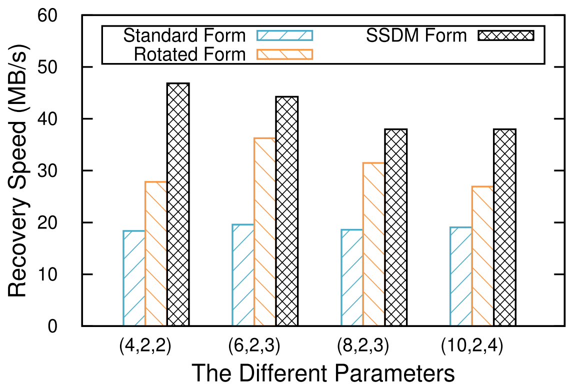


(a) LRC Code.

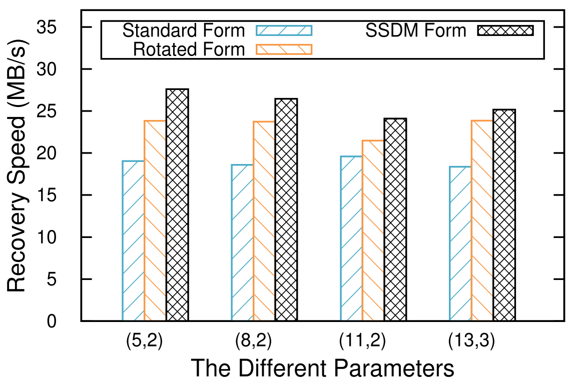


(b) Cauchy RS Code.

Fig. 5 Recovery balance for the tested erasure codes with different forms.



(a) LRC Code.



(b) Cauchy RS Code.

Fig. 6 The recovery speed for the tested erasure codes with different forms.

X5472 processor and 12GB memory. The operation system is SUSE with Linux fs91 3.2.16.

We consider LRC codes and Cauchy RS codes with the same forms and parameters referred in Sect. 4. For each form with different parameters, we evaluate the real recovery speed under each potential data disk failure, and show the average speed in Fig. 6. For standard forms, the recovery speed for both LRC codes and Cauchy RS codes are similar, because the recovery balance is stabilized. For rotated forms and SSDM forms, the recovery speed varies in proportion with the recovery balance referred in Sect. 4. In statistics, SSDM forms gain 20.6% to 68.4% higher recovery speed than that with rotated forms for LRC codes, and achieve 5.9% to 16.1% higher recovery speed than that with rotated forms for Cauchy RS codes, respectively. The results match analysis well and illustrate that SSDM could effectively improve the performance on single disk failure recoveries.

6. Conclusion

Single disk failure recovery problem is very important in today's storage systems. In this paper, we propose a new strip-switched method termed SSDM, in order to further optimize the single disk recovery performance. SSDM utilizes the concept that strips of each stripe of erasure codes could be switched, and uses the simulated annealing algorithm to search for the proper strip-deployment on the stack level to balance the read accesses among local disks, in order to utilize the parallelism of disk reads to optimize the recovery performance. The analysis and experiment results show that, SSDM provides very good performance on single disk failure recoveries for erasure coded storage systems.

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, 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.F. ul Haq, M.I. ul Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas, "Windows azure system: A highly available cloud storage service with strong consistency," Proc. ACM SOSP'11, 2011.

[3] 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.

[4] Z. Shen, J. Shu, P. Lee, and Y. Fu, "Seek-Efficient I/O Optimization in Single Failure Recovery for XOR-Coded Storage Systems," IEEE Trans. Parallel Distrib. Syst., vol.28, no.3, pp.877–890, 2017.

[5] X. Luo and J. Shu, "Load-Balanced Recovery Schemes for Single-disk Failure in Storage Systems with Any Erasure Code," Proc. IEEE ICPP'13, 2013.

[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, 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] S. Xu, R. Li, P.P.C. Lee, Y. Zhu, L. Xiang, Y. Xu, and J.C.S. Lui, "Single disk falure recovery for X-code-based parallel storage systems," IEEE Trans. Comput., Jan. 2013.

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

[10] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in Windows Azure Storage," Proc. USENIX ATC'12, 2012.

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

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