ATLANTIS PRESS

Advances in Engineering Research, volume 149

2nd International Conference on Mechanical, Electronic, Control and Automation Engineering (MECAE 2018)

An Erasure Code-based Approach to Improve Data Recovery and Update Capability

Hang Zhou, Yahui Yang, Weiping Li

Peking University, Beijing 100871, China

Keywords: Erasure code, Algebraic signature, Shell choose, Dynamic update, Monitor.

Abstract. The growing demand of cloud storage service has raised a great concern to service providers aiming at high service reliability and usability. In this work, we proposed an erasure code-based approach SEDP to improve the capability of data recovery and dynamic update for cloud storage. The coding scheme adopts the main ideology of Shell Sort to maximize the physical distance of data blocks in each coding group which helps recover damaged data blocks in extreme damage conditions. The concept of data page is introduced to simplify and standardize dynamic update procedure. We then designed a smart monitor to patch the cached data pages to the corresponding data blocks to maintain consistency. The patching work is based on two different granularities, time period and modification threshold. These two key parameters are tuned by the monitor according to the information it collects. The prototype is experimentally evaluated in simulated scenarios, focusing on its performance compared with related works and how the monitor improves the performance as the system runs continuously.

1. Introduction

Since increasing number of people put their sensitive data files on the cloud, cloud service providers (CSP) are confronted with challenge in terms of reliability and usability. To achieve reliability, firstly the service providers need to provide proofs that the stored data is not modified or partially deleted. Since once the data is uploaded to the cloud, data owners lose direct control of them. One solution is called Provable Data Possession (PDP) introduced by Atenies et al [1], which uses challenge-response protocol to do the verification. Based on this scheme there are different variations [2] [3]. Secondly, the data should be stored in a recoverable way since there are alarming trends in disk failure rate [14] [15]. A classic method is storing backups in different data nodes using allocation algorithm like CRUSH [4], which is applied in Ceph. Another popular method is using erasure code to encode the file and store the parity information, which can save storage and network resources. It defers in thousand ways when apply the erasure code into real application, such as RS code [5] or other MDS code STAR [18].

The files on the cloud are not immutable. For example, Individual users will frequently edit their remote documents online, while enterprise users may periodically reorganize their server logs on the cloud. To update data files, common practice is to split an update request into delete-and-reupload. But this solution is a waste of computing resource in erasure code-based system since the whole data file needs to be partitioned and encoded again. Data files being formed into high recoverable structured usually leads to large parity blocks and low update efficiency. System needs to tradeoff between factors such as algorithm complexity and storage expense.

In this work, we designed an approach called SEDP, it uses erasure code-based Shell encoder to improve data recovery capability. The Shell encoder chooses data blocks at regular distances, similar to the way the Shell sort chooses elements. Then we use data page to transform dynamic update request into standard procedure. Since the data pages are cached like patches, a smart monitor is designed to do periodical patching work and tune the system parameters. The proposed approach aims to tackle challenges such as recovering data blocks in extreme damage conditions and reducing the I/O and CPU overhead during update process.

2. Literature Review

2.1 Erasure Code & Algebraic Signature

Erasure code [6] [7] is also known as the FEC code (Forward Error Correcting codes). It requires much computing resources but greatly reduces the redundancy of the system and saves a lot of storage space compared with common replication schemes. The basic idea of erasure code is to encode k data blocks and get m parity blocks. Within the n=k+m blocks, as long as no more than m of them damaged, all the original ones can be recovered through the rest of the t(t>=k) using reconstruction algorithm. When t=k, the code has the property of MDS (Maximum short Separable), then it is called MDS code. The efficiency of erasure code is usually defined as e=n/k (e>1), as he grows, data recovery ability becomes stronger due to more parity blocks. RS Code (Reed-Solomon Code) is a typical MDS codes which needs to be added and multiplied on Galois Field GF (2w). RS Code can be classified into Vandermonde RS Code [5] and Cauchy RS Code [8] according to different generator matrix. The Vandermonde RS Code uses discrete logarithm to inverse the generator matrix which causes large computational overhead. Cauchy RS Code uses Cauchy matrix to turn multiply operation in GF (2w) into binary multiply only involving XORs, which greatly reduce the computing overhead.

Litwin W, Schwarz T [9] point out that algebraic signature can be applied into erasure code-based system to achieve PDP. An algebraic signature is defined assig_α, it is a hash function which can compress big files into strings like common MD5 functions do. Given a string $x_0, x_1, \dots, x_{n-1}$, its algebraic signature can be represented as Eq.1. α is a prime number in GF(2w).

$$\text{sig}_\alpha(x_0, x_1, \dots, x_{n-1}) = \sum_{v=0}^{n-1} x_v \cdot \alpha^v \quad (1)$$

Algebraic signature has the properties of homomorphism and algebraicity. These two properties can be used to prove that the sum of each data block's algebraic signature equals to the algebraic signature of the sum of data blocks. Represented $\text{assig}_\alpha(X \oplus Y) = \text{sig}_\alpha(X) \oplus \text{sig}_\alpha(X)$. This calculation is simple and quick using only XOR operations without complex logic operations.

In [10], challenges are sent to get random data bytes from the file and calculated its parity signature by the verifier. Then the signature will be compared with signature of its corresponding position on the stored parity blocks, showing whether the file is modified. Since the hash function and secret key (generator matrix) cannot be reversely cracked, the storage server cannot fake signatures thus the verification is safe.

2.2 Coding Schemes

A few erasure code-based coding schemes are designed which aims at achieving optimal encoding, updating and decoding performance. WEAVER Codes [17] can reduce the read overhead by using twice as much storage spaces as the data collection itself. To reduce the storage overhead and maintain access efficiency at the same time, Pyramid Codes [12] proposed by Microsoft Research uses local and global redundancy and multi-hierarchical extension. It is a more flexible scheme compared with WEAVER Codes. A MDS array erasure code called DA-codes [13] is designed to improve recovery capability when extreme damage condition happens, it adopts two types of parity placed in two directions. DA-codes can tolerate up to two disk failures while traditional RAID [16] (Redundant Arrays of Inexpensive Disk) and its striping techniques can not satisfy. The common point is that all of the schemes are making tradeoffs between storage space and access efficiency in reliable data storage systems

A map-based dynamic data integrity verification and recovery scheme (MB-DDIVR) [11] in our previous work is an approach which can check data integrity and recover the damaged blocks in two granularities. It followed PDP thus can prevent multiple cloud servers from colluding to fabricate consistent signatures, the verification work is delegated to a third party assistant (TPA). Cauchy RS code and algebraic signature are combined in MB-DDIVR. Open-sourced work including Erasure coding libraries and fast Galois field arithmetic library [19] are applied to help implement two key functions in MB-DDIVR. a.Encode data blocks and decode(recover) them. b.Check the integrity of coding groups. These two techniques are also applied to this work. In the previous model evaluation, MB-DDIVR shows low data recovery capability in some extreme damage condition. Another defect is that it cost great I/O during update process. SEDP in this paper is designed as a complementary approach to solve these problems.

3. Methodology

We present our methodology in this section. The Shell encoder and the monitor are deployed on TPA and all the blocks or pages generated by Shell encoder will be uploaded to the storage cloud through standard interface.

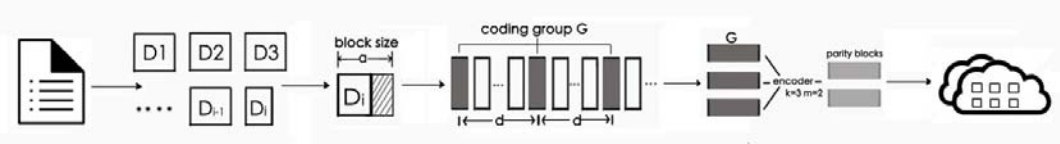

Figure 1. Data encoding process

3.1 Data Encode and Decode

The file is firstly partitioned into data blocks of the same size. Then the last unsatisfied block may be filled with blank bytes because coding phase requires byte alignment. The blocks are then grouped and encoded using RS erasure code. Taking 52-encode for example, each group contains 5 data blocks and 2 parity block. Each group will be uploaded separately thus they may be distributed on different disks. Fig.1 shows the overview of the encode phase.

In MB-DDIVR, the encoding process is also based on groups and it focus on designing data structure within the group. But it ignores the way to group the blocks thus successive blocks after partition phase will be formed as one group. In some file system such as linux ext* file system, when the data blocks are written to the disks, they may not be contiguous with each other. But as the file size grows, their distribution on the hard disks tends to be successive on the whole. When magnetic tracks are scratched on one disk or successive disks are damaged, the damaged blocks will be centrally distributed, resulting in a reduction in recovery capability because there is a high probability the damaged blocks belong to the same coding group. Therefore, the chosen data blocks in each group should be decentralized as much as possible. The Shell encoder adopts the main ideology of Shell sort, aiming at maximize the physical distance of data blocks in each encoding group. For example, the file is partitioned into i blocks denoted by $(b_0, b_1, b_2, \dots, b_{i-1})$ and we use k-m scheme to encode them which means a coding group contains k data blocks and m parity blocks. Then j blocks are damaged successively denoted by $(b_0, b_1, b_2, \dots, b_{j-1})$. If the coding process goes normal linearly, then $(b_{j-(j \bmod k)}, b_{j-(j \bmod k)+1}, \dots, b_{j-1})$ totals to j mod k blocks can be recovered when j mod k > m, because these blocks belong to the same coding group which has less than m damaged blocks. When j mod k <= m, no blocks can be recovered. The number of recoverable blocks denoted by N can be calculated as follows:

$$N_1(\text{recovered}) = \begin{cases} j \bmod k & , j \bmod k \le m \\ 0 & , j \bmod k > m \end{cases} \quad (2)$$

Then if the coding blocks are chosen in a distance d=i/k, in the best case when j<=dm, all of the damaged blocks can be recovered. When dm<j<dm+d, total to j-dm groups $\{(b_0, b_d, \dots b_{i-d}), (b_1, b_{1+d}, \dots b_{1+i-d}), \dots, (b_{j-dm-1}, b_{j-dm-1+d}, \dots b_{j-dm-1+i-d})\}$ each has m+1 blocks damaged inside thus they cannot be recovered. N is calculated according to Eq.3.

$$N_2 = \begin{cases} j & , j \le dm \\ dm^2 + dm - jm & , dm < j < dm + d \\ 0 & , j \ge dm + d \end{cases} \quad (3)$$

Taking i=100, k=5, m=2, j∈[1,100], we get Avg(N1)=0.4, Avg(N2)=12, the recover ability increases 29 times by using shell encoder.

To check the integrity, the TPA sends challenges to the server to get random data bytes and calculate the algebraic signature. The result will be compared with the data bytes on the corresponding parity blocks. The verification work goes iteratively on each coding group and the damaged groups will be sent to the decoder to recover the original ones. Algorithm 1 shows the detail procedure.

3.2 Dynamic Data Update

To meet the need for dynamic data update. We adopt linear data pages instead of tree-based structure or map-based structure in MB-DDIVR. In some DB system such as ORACLE and MYSQL,

data page is a basic IO unit which greatly improves CRUD efficiency. With data pages, complex data structures can be realized such as different types of index in MYSQL storage engine. Data pages in this system are files of different size, they are regarded as patches of the original data blocks. With these pages, the update procedures could be accelerated, standardized and simplified. CRUD operations could all be regarded as replacement and insertion of pages, defined as $(P_{length}^{offset}, Type)$. For example, $(P_{1\,n_1}^{o_1}, D)$ represents the replacement of a n1-byte page P_1 with a blank page, it equals to delete n1 bytes data from the specified offset o_1. $(P_{2\,n_2}^{o_2}, I)$ represents the insertion of a n2-byte page P_2 behind offset o_2, it equals to insert n_2 bytes behind offset o_2. The data pages may also suffer from damaging because they are stored as cache files on the hard disk. Thus they are encoded with their neighbors on the file and generate the parity pages to ensure recover capability. Given $P_{2\,n_2}^{o_2}$, a new page $P_{3\,n_2}^{o_2-n_2}$ will be cut out from the file and sent together with P_2 to do encoding. Fig.2 shows the overview of the update phase. It uses 21-encode scheme.

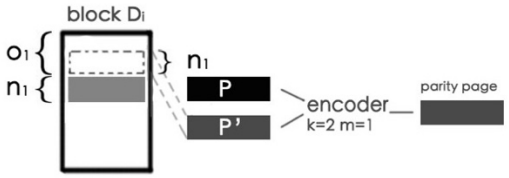

Figure 2. Dynamic update phase

When there comes download request, the cached pages will be replaced or inserted to the specified offset. It is important that the page offsets $O(o_0, o_1, o_2, \ldots, o_{n-1})$ of the same block need to be sorted in reverse order and handled in sequence. The reason is once $\left(P_{1\,n_1}^{o_1}, D\right)$ is handled, $o_2 (o_2 > o_1)$ in $(P_{2\,n_2}^{o_2}, I)$ expires and needs to be recalculated because the last operation changes the data length ahead of it.

3.3 Smart Monitor

"A Thousand Gapping Wounds" can be used to describe the status of the blocks during the update phase. In a common DB transaction, modified pages will be cached as files instead of writing to DB instantly, handling pages in this scheme will also be delayed due to limited computing resource. As the data blocks are updated frequently, several cached data pages may point to different indexes of a same data block at the same time. At that point if a download request comes, the assembling time will be relatively long. Hence the monitor is designed to solve this problem. It periodically patches the data pages to corresponding blocks to make the file consistent with users' operations. As long as the patching work is done, the monitor redo the encoding. This patching operation will be carried out based on two different granularities, time period α and modification amount threshold β. These are two key parameters in this scheme and they are tuned by the monitor according to information it collects. Table I specifies the relation between parameters and environment variables.

Function G(x) in Table I is a normalization function which calculates the variation of the inside parameter and normalize it to 0.5~2, which means G(x)'s result will at most double or halve the parameters it influences. G(x) prevents the parameters from changing too rapidly, ensuring the system have a smooth transition.

Table.1 ensuring the system have a smooth transition.

Environment variable	Patching Parameter	Relation
P_1 update period P_2 download period P_3 response time S_1 data page size	α time period D_α default period	$\alpha = \begin{cases} D_\alpha \cdot G\left(\frac{P_1 P_2}{S_2 P_3}\right), \alpha \geq D_\alpha \\ D_\alpha \qquad\qquad\quad , \alpha < D_\alpha \end{cases}$
	β modification amount threshold D_β default threshold	$\beta = \begin{cases} D_\beta \cdot G\left(\frac{1}{S_2 P_3}\right), \beta \geq D_\alpha \\ D_\beta \qquad\qquad\quad , \beta < D_\alpha \end{cases}$

4. Evaluation

In order to evaluate the performance of the proposed model (SEDP), we have conducted experiments to compare with other models. MB-DDIVR is the only approach we currently found which supports data recovery and handles dynamic update request at the same time. Thus we compare SEDP with MB-DDIVR in 3 different aspects including data recovery performance, data update performance and data download (read) performance. Before the samples are taken, to avoid interference in the time measurements, irrelevant top 5 CPU-consuming processes are killed using Linux command 'ps aux | grep –v system | sort –nrk 3,3| head –n 5 | xargs kill –s 9' and then the model is restarted. System I/O buffers are flushed using Linux command 'sync'. We then wrote Linux shell scripts to automatically run the simulation for several successive times in a single thread and calculate the average value.

Here is our server and network configuration. CentOS 6.5 64-bit, 2G memory, single core 2.20GHz CPU, average 1.2MB/s download speed and 0.23MB/s upload speed.

4.1 Data Recovery Ability

Block size and coding schema will make difference to the experiments result described in the previous sections. To control the variables in the established experimental model, preparatory experiment is performed. Fig 3 compares three different encoding schema and it plots the relation between time overhead and block size ratio in the data encoding process. As the block size ratio increases, less coding group will be generated during the encoding. Thus there exists less file binding handles, which slightly helps save time overhead but will not make big difference to the overall result. To make the following experiments representative and typical, the block size ratio is set to 1% and 4-3 encoding schema is chosen. These parameters are at intermediate level according to Fig 3.

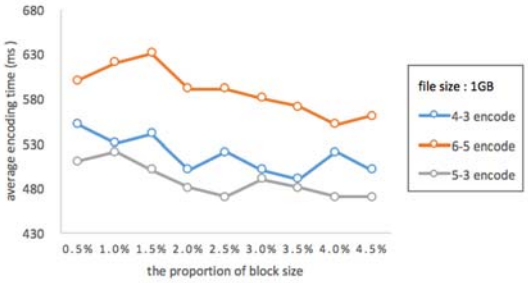

Figure 3. Data recovery performance

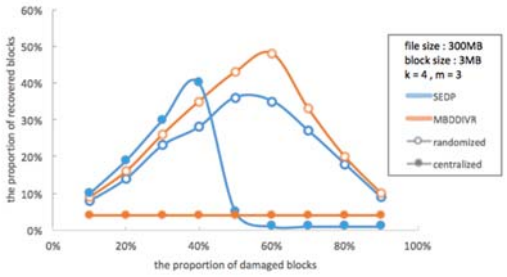

Figure 4. Data recovery ability

To compare the recovery capability of SEDP with MB-DDIVR in different damage conditions, we simulate two types of damage distribution in experiment 1, centralized and randomized. Centrally distributed damage means contiguous data blocks are deleted or modified while randomly distributed means the positions of damaged blocks are independent. The damage work is performed by the automatic shell script. Fig.4 plots the proportion of recovered blocks as the damage rate increases. We can observe that the recover percentage of SEDP is 24.1 %(0.29-0.22/0.29) lower compared with MB-DDIVR in random damage condition. The reason is MB-DDIVR uses map-based storage table to store more parity blocks for each group. However, when the damage is centralized, SEDP can recover all the blocks as long as the damage proportion is lower than 42.8% which can be regarded as damage tolerance. While under this condition MB-DDIVR can only recover average 4% of the blocks, which corresponds to our derivation of Eq.2 and Eq.3.

Fig.5 plots the average recovery time when the damage percentage increases. SEDP has shorter recover time while MB-DDIVR spends additional time to maintain its map-based block structure. During integrity check and data recovery process on TPA described in Algorithm1, the main overhead is network I/O due to data uploading and downloading, which is inevitable. We only record the time cost by the key steps, from the beginning of integrity check to the end of recovery.

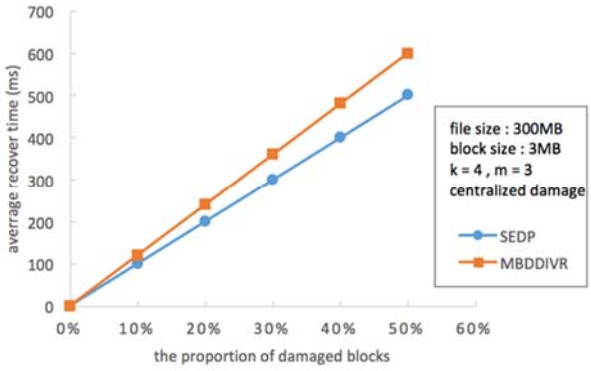

Figure 5. Data recovery performance

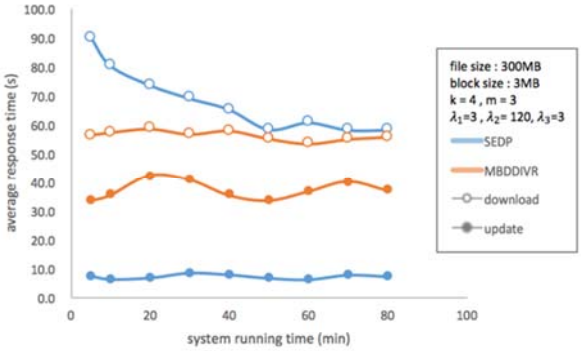

Figure 6. Data update download performance

4.2 Update & Download Performance

To observe the update and download performance in SEDP and how it is effected by the monitor, we make different requests and continuously sent them to the system and then record the average response time. It assumes that the period of the requests P_1, P_2 and the size of update requests S_1 have Poinsson distribution with meanλ_i, denoted as

$$P_1 \sim \text{Poinsson}(\lambda_1), P_2 \sim \text{Poinsson}(\lambda_2), S_1 \sim \text{Poinsson}(\lambda_3) \quad (4)$$

Each time the current request is sent, the automatic script calculates the size of next request and generate the request body by random characters using Linux command 'cat', then the script calculate next interval and reset the timer. Fig.6 shows the simulation result. It is obvious that SEDP has high update efficiency, which means the cached data pages greatly reduce the network I/O and disk I/O compared to block-size update in MB-DDIVR. In the aspect of download performance, SEDP has a poor performance 20.3 %(68.2-56.1/68.2) lower in the initial stage. But the average download time gradually goes down and is finally levelling out at 58.2s. When the average response time tends to be stable around after 80min running time, we record the environment variables and the system parameters. $P_1, P_{2,}, S_1$ Calculated by monitor approximate to $\lambda_1, \lambda_2, \lambda_3$(3s, 120s, 3MB), the threshold β and patching period α are tuned to 42.3MB and 72.5s. The comprehensive result shows that update performance could be improved 5.3(27.1/7.0) times at the cost of 20.3% download performance decrease.

5. Conclusion and Future Work

In this paper, we have proposed an effective model for storage systems, SEDP, which helps cloud providers have an idea of how to store their data in a recoverable way. We focus on how to improve the recover capability and update the data effectively. The model is designed to be "Plug and Play", which can be applied to read-only system. By turning on the monitor module, it can also handle with dynamic update request without making big changes inside the structure.

Compared with related works in the information security industry and in data storage literature, our model is positioned as a complementary approach. It uses different encoding scheme compared with MB-DDIVR and it adopts data page instead of map-based table to realize dynamic update. By conducting the first experiment, we show that SEDP have better recovery capability when the damage blocks are centrally distributed. The second experiment shows that SEDP is an update-friendly model. It can effectively handle with frequent update request at the cost of little download efficiency loss. Therefore, if files are frequently updated and the update-download ratio is high, SEDP shows a better overall performance. Otherwise MB-DDVIR will still be a good solution.

In future work, we will consider applying different granularities into the shell choosing phase to make the system more flexible and damage tolerable. We also intend to refine the smart monitor by taking more relevant factors into consideration.

6. Appendix

Algorithm 1 Integrity check & data recovery process on TPA

Input: blocksize, filename, k-m encode, checkinglevel
Output: unrecovered blocks D[]

G = getAllGroups(filename);
for each g = $(b_0, \dots, b_{k+m-1}) \in$ G
set coveredBytes = 0
while coveredBytes < blocksize/ checkinglevel **do**
length = randomInt(blocksize/2);
coveredBytes += length
for i $\in \{0, \dots, k-1\}$
$S_1 = S_1 \oplus \text{signature}(b_i, \text{coveredBytes}, \text{length})$
end for
for j $\in \{k, \dots, k+m-1\}$
$S_2 = S_2 \oplus \text{signature}(b_j, \text{coveredBytes}, \text{length})$
end for
if erasureEncode(S_1) != S_2 **then**
if erasureDecodeGroup(g) == 'success'
uploadGroup(g)
else
D.add(g)
break
end if
end while
end for

Algorithm 2 Patching process

Input: file

Initialization**:** modified groups G = []

P = getAllCachedPages(file)
sortByOffsetDesc(P)
for each $p_i \in P$
g_i = getBelongGroup(p_i)
b_i = getCorrespondBlock(p_i, g_i)
patchToBlock(p_i, b_i)
G.add(g_i)
end for
erasureEncodeGroups(G)
uploadGroups(G)
deletePages(P)

References

[1]. Ateniese, G., Burns, R., Curtmola, R., Herring, J., Kissner, L., Peterson, Z., Song, D.: Provable data possession at untrusted stores. In: Proceedings of the 14th ACM Conference on Computer and Communications Security. ACM (2007)

[2]. Erway, C.C., Küpçü, A., Papamanthou, C., Tamassia, R.: Dynamic provable data possession. ACM Trans. Inf. Syst. Secur. (TISSEC) 17(4), 15 (2015)

[3]. Hao, Z., Yu, N.: A multiple-replica remote data possession checking protocol with public verifiability. In: Proceedings of 2nd International Symposium Data, Privacy, E-Commerce, pp. 84–89 (2010)

[4]. S. A. Weil, S. A. Brandt, E. L. Miller and C. Maltzahn, "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data," SC 2006 Conference, Proceedings of the ACM/IEEE, Tampa, FL, 2006, pp. 31-31.

[5]. Reed I S, Solomon G. Polynomial Codes Over Certain Finite Fields [J]. Journal of the Society for Industrial & Applied Mathematics, 1960, 8(2):300-304

[6]. Plank J S. Erasure Codes for Storage Applications. Tutorial Slides Presented at FAST-2005: the 4th USENIX Conference on File and Storage Technologies, 2005.

[7]. Sloane, N. J A. The theory of error correcting codes / [M]. North-Holland Pub. Co., 1978.

[8]. Roth R M, Lempel A. On MDS codes via Cauchy matrices [J]. IEEE Transactions on Information Theory, 1989, 35(6):1314-1319.

[9]. Litwin W, Schwarz T. Algebraic signatures for scalable distributed data structures[C]//Data Engineering, 2004 Proceedings 20th International Conference on. IEEE, 2004: 412-423.

[10]. Schwarz, T.S.J., Miller, and E.L.: Store, forget, and check: using algebraic signatures to check remotely administered storage. In: 26th IEEE International Conference on Distributed Computing Systems, 2006. ICDCS 2006. IEEE (2006)

[11]. Sun zizhou, Yang yahui, Shen qingni, Wu zhonghai, Li xiaochen. MB-DDIVR: A map-based dynamic data integrity verification and recovery scheme in cloud storage [C]//Proceeding of 17th International Conference on Information and Communications Security. Springer, 2015:335-345.

[12]. Huang C, Chen M, Li J. Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems [J]. Acm Transactions on Storage, 2007, 9(1):79-86.

[13]. R. Hu, G. Liu and J. Jiang, "An Efficient Coding Scheme for Tolerating Double Disk Failures," High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on, Melbourne, VIC, 2010, pp. 707-712.

[14]. E. Pinheiro, W.-D. Weber, and L. A. Barroso, “Failure trends in a large disk drive population,” in FAST-2007: 5th USENIX Conference on File and Storage Technologies. USENIX Association, 2007.

[15]. J. Elerath, “Hard-disk drives: The good, the bad, and the ugly,” ACM Queue, 2009

[16]. D. Patterson, G. Gibson, and R. Katz, “A case for redundant arrays ofinexpensive disks (RAID)”. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pgs. 109-116,1988.

[17]. J. L. Hafner, “WEAVER Codes: Highly Fault Tolerant Erasure Codes for Storage Systems”, the 4th USENIX Conference on File and Storage Technolgoies (FAST 2005), San Francisco, CA, Dec. 2005

[18]. C. Huang, and L. Xu, “STAR: an Efficient Coding Scheme for Correcting Triple Storage Node Failures”, the 4th USENIX Conference on File and Storage Technolgoies (FAST 2005), San Francisco, CA, Dec. 2005.

[19]. James S. Plank, Kevin M. Greenan, “Jerasure: Erasure Coding Library” http://jerasure.org, http://lab.jerasure.org/jerasure/jerasure

[20]. James S. Plank, “Fast Galois Field Arithmetic Library in C/C++”, http://web.eecs.utk.edu/~plank/plank/papers/CS-07-593