A Cascading Latin Scheme to Tolerate Double Disk Failures in RAID Architectures[1]

Lin Sheng, Wang Gang, Liu Xiaoguang, Liu Jing

(*Nankai-Baidu Joint Lab, College of Information and Technical Science, Nankai University, Tianjin 300071, China*)

Abstract: In recent years, a lot of XOR-based coding schemes have been developed to tolerate double disk failures in Redundant Array of Independent Disks(RAID) architectures, such as EVENODD-code, X-code, B-code and BG-HEDP. Despite those researches, the decades-old strategy of Reed-Solomon (RS) code remains the only popular space-optimal (Maximum Distance Separable) code for all but the smallest storage systems. The reason is that all those XOR-based schemes are too difficult to be implemented, it mainly because the Coding-Circle of those codes vary with the number of disks. By contrast, the Coding-Circle of RS code is a constant. In order to solve this problem, we develop a new MDS code named Latin code and a cascading scheme based on Latin code. The cascading Latin scheme is a nearly MDS code (with only one or two more parity disks compared with the MDS ones). However, it keeps the Coding-Circle of the basic Latin code (i.e. a constant) and the low encoding/decoding complexity similar to other parity array codes.

Key words: 2-erasure code； RAID(Redundant Array of Independent Disks)；Latin square

I. Introduction

With the increasing requirements of disk systems, very large storage systems have to face the problem of two or more disks failing at the same time. However, there is no easy way to resolve that just like what the single-failure-tolerable RAID system does. Therefore, researches on erasure-coding have blossomed in recent years. The Reed-Solomon (RS) code [1, 2, 3], which introduced from coding theory, can meet the requirements and becomes popular in these years. Actually, RS code has been used to build some massive storage systems in real life. However, the RS code's shortcomings are also obvious. Following the coding theory, RS code is a general scheme to solve t-erasure-correcting problem, and its computational complexity is much higher than that of XOR-based codes.

In recent years, many XOR-based 2-erasure-correcting codes have been designed, such as EVENODD[4-7], X-code[8], B-code[9] , BG-HEDP[10-13] and Liberation-code[14] etc. All of the codes have reached the Singleton bound, that is to say they are space-optimal codes. Compared with RS code, the decoding complexity of these codes is much lower. However, it is very interesting that RS code is the only 2-erasure-correcting code which is widely used by industrial community. RS code became the winner because it is easier to be implemented than other codes.

---

[1] Manuscript received date: April 19,2009 ; revised date: December 2,2009.
This work was supported in part by the National High Technology Research and Development Program of China (2008AA01Z401), the National Science Foundation of China (60903028), Doctoral Fund of Ministry of Education of China (20070055054), and Science and Technology Development Plan of Tianjin (08JCYBJC13000).
Communication author: Lin Sheng, born in 1973, male, Doctor. Nankai-Baidu Joint Lab, College of Information and Technical Science, Nankai University, Tianjin 300071. shshsh.0510@gmail.com.

In the following sections of this paper, we discussed the details of the codes mentioned above and gave a careful analysis about the reasons of their difficulties to be implemented. The contributions of this paper can be described as follows:

Firstly, we gave some new criteria for the code availability. Secondly, we designed a new 2-erasure-correcting MDS code named Latin code. Finally, based on Latin code, we developed an appropriate scheme which can meet all new criteria.

II. Analysis of Coding Availability

The XOR-based coding schemes, such as EVENODD code, have excellent characters to meet the requirements of massive storage system. We sum up them as below:

(1) Almost all of them are space-optimal coding schemes. It means that only two parity disks are required to tolerate 2-erasure in the system.

(2) They have optimal update penalty. It means that when a data disk is updated, only two corresponding parity disks need to be updated.

(3) Compared with other RS-like coding schemes, the encoding/decoding complexity is much lower [4,5].

However, the XOR-based coding schemes have some shortcomings which prevent them from being applied.

(1) Some schemes, such as EVENODD and X-code, require the number of disks must be a prime number. Other schemes, such as B-code and BG-HEDP, are not so strict, but can not be applied to any array sizes.

(2) We call the average amount of data units on every disk in a parity stripe as the coding cycle of a code scheme. The coding cycle of XOR-based coding schemes is linear with a number of disks. It means that the XOR-based coding schemes are not a general method for massive storage systems. Because the cycle is related to the number of disks, users must pay more attention to setting the coding cycle according to capacity and the number of disks. On the contrary, RS code is a general code (For a given GF(2^n), its coding cycle can be seen as a constant), therefore , it can be implemented easily.

(3) The extensibility of XOR-based coding schemes is weak. The system is difficult to be extended after having been built. The reason lies in the changing coding cycle.

If the number of disks $n<259$, using Fermat prime numbers, a feasible scheme was presented for EVENODD code in Ref [5]. The coding cycle in this scheme is 256. It is easy to be mapped to physical storage devices (a coding cycle apropos construct 32 bytes, namely 256 bits). For storage systems, 256 devices are enough for most cases. Unfortunately, the next Fermat prime number is 65537, which is too big for coding cycle. To choose other primes, it is very difficult the keep the coding cycle divided exactly by the usually disk Read/Write unit size (for instance 4k bytes). Therefore, while n is bigger than 259, it is a challenge to find a feasible prime number.

Full-2 code [10, 15] is also a general code (its coding cycle is a constant). But its redundancy is $O(n^{1/2})$, which is too high (the optimal value is only 2).

Maximal projective code (Hamming code) has optimal redundancy when we fixed the coding cycle to 1, but its update penalty is too high (about $n/2$ on average, the optimal is 2).

B-code, with an exact 2 update penalty and a space-optimal character, has a various coding cycle (linear to the number of disks) that makes it rather hard to be implemented.

In a word, the tradeoff among code length, coding cycle, update penalty and redundancy must be considered when a coding scheme is designed.

From the comparison in Tab 1, we can conclude why the RS code is so popular. It is the easiest one to be implemented for an arbitrary number of disks. A constant coding cycle is a very important property since it determines the way how a scheme be mapped into a physical system. Compared with the implementing difficulty, the redundancy may be a minor problem.

Tab1 Comparison between 2-erasure-correcting codes

Coding scheme	Coding cycle (*C*) Universal	Disk number (*N*)	encoding complexity	Redundancy
RS	For the RS code based on F_8, *C*=8, each cycle forms one-byte.	$N<2^C$ When *C*=8, *N*<256	High	2/*N*
EVENODD	C=N-1, *N* should be a prime number, not universal	*N*<=*C*+3	Low	2/*N*
X-Code	*C*=*N*-2, N should be a prime number	*N*<=*C*+4	Low	2/*N*
B-code	*C*=*N*/2	*N*<=*C*/2	Low	2/*N*
BG-HEDP	*C*=*N*	*N*<=*C*+3	Low	2/*N*
Full-2	*C*=1	No limit	Low	$N^{1/2}/N$
2D array	*C*=1	No limit	Low	$N^{1/2}/N$
Hamming code	*C*=1	No limit	Low	lg(*N*)/*N*

Based on the above analysis, we give our new criteria for a “Good” 2-erasure-correcting scheme, and give more priority to implementing difficulty:

(1) The Coding cycle keeps a constant number (it makes the scheme easier to be implemented);

(2) Redundancy <= O(lg*N*) for *N* data disks (Be the same with Hamming code, it is the best possibility);

(3) A constant update penalty (Compared with being fixed to 2, it has been loosed).

In next two sections, we focus on constructing a code scheme to achieve all these 3 criteria.

III. Latin -code

To achieve the metrics proposed in Section II, firstly, we introduce a new MDS code: Latin-code.

Definition 1 a Latin square consists of *N* permutations of {*1*, *2*, …, *N*} which are arranged in such a way that no row and column contains the same number twice [16,17].

We denote the permutation in the *i*-th column by σ_r, the symbol in Row *i*, Column *j* by $\sigma_j(i)$. We define $\sigma_{r,s}=\sigma_r\sigma_s^{-1}$, that is, the cycle pattern formed by Column *r* and Column *s*.

Definition 2 A *Latin square* is column-Hamiltonian if each pair of columns forms a single cycle, that is, $\sigma_{r,s}$ contains a single cycle [17].

A column-Hamiltonian Latin square of order 9 - *L*9 is given in Fig. 1.

Suppose *L* is a given reduced (the first row and the first column are in natural order)

column-Hamilton Latin square, then we can construct a new double disk failures tolerable system based on it.

Algorithm 1:

Step1 Map each column to a disk, and each symbol in the square represents a stripe unit except the last row. We regard the last row as a dummy row and suppose the dummy data unit is always zero.

Step2 Add two parity check disks named *P*, *Q* to the system. Suppose $D_{j,k}$ denotes the *k*-th stripe unit in the *j*-th data disk and P_i and Q_i denotes the *i*-th stripe units in the first and the second parity check disk respectively, then P_i and Q_I are calculated by:

$$P_i = \bigoplus_{1\le j\le n} D_{j,i} \quad , \quad 1\le i\le n \tag{1}$$

$$Q_i = (\bigoplus_{1\le j\le n,\sigma_j(k)=i} D_{j,k}) \oplus S \quad , \quad 1\le i\le n \tag{2}$$

$$S = (\bigoplus_{1\le j\le n,\sigma_j(k)=n} D_{j,k}) \tag{3}$$

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 2 & 4 & 8 & 9 & 3 & 5 & 1 & 7 & 6 \\ 3 & 1 & 9 & 2 & 8 & 7 & 5 & 6 & 4 \\ 4 & 5 & 2 & 3 & 1 & 8 & 6 & 9 & 7 \\ 5 & 7 & 4 & 1 & 6 & 9 & 8 & 3 & 2 \\ 6 & 9 & 5 & 8 & 7 & 4 & 2 & 1 & 3 \\ 7 & 8 & 6 & 5 & 9 & 2 & 3 & 4 & 1 \\ 8 & 6 & 1 & 7 & 4 & 3 & 9 & 2 & 5 \\ 9 & 3 & 7 & 6 & 2 & 1 & 4 & 5 & 8 \end{bmatrix}$$

Fig 1 A column-Hamiltonian Latin square of order 9 - *L*9

It is obvious that *P* is the horizontal parity check of each row. *S* is the sum of all symbols labeled "*n*" in the Latin square. The *i*-th symbol in *Q* is the sum of *S* and all symbols labeled "*i*". We call this check disk the Latin parity disk, the parity groups on it the Latin parity groups, and the parity units the Latin parity units.

Theorem 1 System constructed by Algorithm 1 can tolerate any double disk failure.

Proof: Without loss of generality, suppose the *i*-th and the *j*-th disks fail ($1\le i<j\le n$). We can deduce that starting from a dummy unit we can recalculate each error data unit step by step.

Case 1 Suppose two data disks fail. From *j*, since the last unit is the dummy one, there is at most one unit failed ($D_{i,a}$) in the Latin parity group $\sigma_j(n)$ (note that $\sigma_j(n)\neq n$). So $D_{i,a}$ can be reconstructed. Then we can reconstruct $D_{j,a}$ by P_a. So the Latin parity group $\sigma_j(a)$ contains only one failed unit now, we can reconstruct it, and so on. Note that we can reconstruct *S* by XORing all units in the two parity disks, and constructing algorithm breaks the single cycle $\sigma_{i,j}$ into paths. Therefore ,this zigzag way can reconstruct all failed data units step by step. Fig.2 shows an example.

Case 2 Suppose a data disk *i* and P_1 fail. If *i*=1, *S* is calculated first; otherwise *S* is calculated by

$$S = (\bigoplus_{1\le j\le n, j\neq i,\sigma_j(k)=\sigma_i(n)} D_{j,k}) \oplus Q_{\sigma_i(n)}$$

According to the basic properties of Latin square, it is clear that each Latin parity group only includes one single failed data unit, therefore all the failed units in disk *i* can be reconstructed by *Q* and *S*. Then we can recalculate *P* by Eq. (1).

Case 3 Suppose a data disk *i* and the Latin parity disk *Q* fail, disk *i* is reconstructed through *P* first, and then *Q* is recalculated by Eq. (2).

Case 4 Suppose the two check disks fail, decoding equals to encoding.

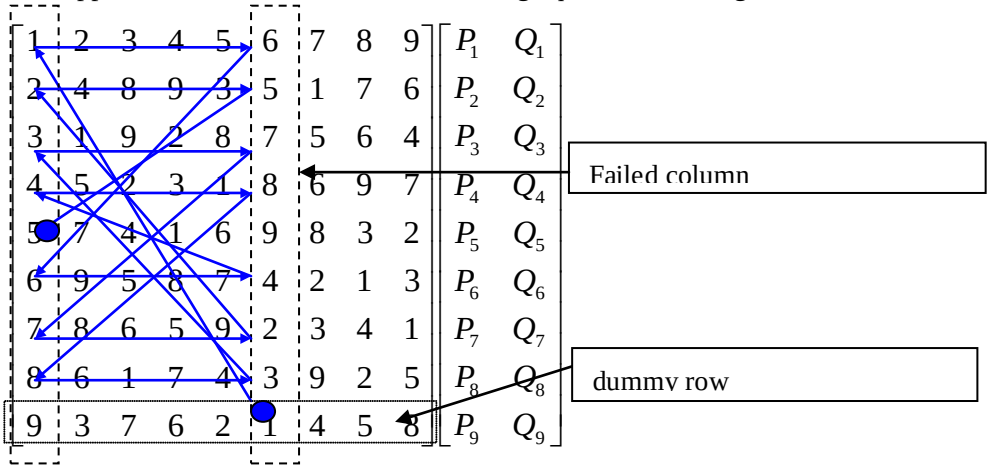

Fig. 2 Decoding of Latin-code

We can see that the decoding algorithm is similar to that of the EVENODD. In fact, when *n* is a prime number, Latin Code is just EVENODD. That is, EVENODD is a special case of Latin Code. According Ref.[16,17], there is a bijection between column-Hamilton Latin squares and the P1Fs (Perfect 1-Factorizations) of complete bipartite graphs. Moreover, we can construct a P1F of the complete bipartite graph $K_{n,n}$ through a P1F of the complete graph K_{n+1} [16]. There is a long history and widely believed conjecture in graph theory field: every complete graph with an even number of vertices has a P1F [16]. So Latin code exists for all odd number if this conjecture holds. Latin code is similar to BG-HEDP and PIHLatin code [11-13] except that the latter two preserve *S* instead of XORing it into all other Latin parity units. A common limit of these three coding schemes is that no theory can guarantee the existence of them for an arbitrary disk number. Especially for some large non-prime number, it is very hard to find a proper column-Hamilton Latin square. Therefore, a system developer can not use those schemes directly and freely. Although horizontal shortening [11] alleviates this problem, it leads to bad encoding and decoding performance. However, we will show that we can get a good coding scheme for arbitrary disk number based on a concrete column-Hamilton Latin square. Specially, the Latin code based on *L*9 has a coding cycle 8. If a symbol represent one bit, then a cycle form nicely a byte which is highly universal.

Now, the remaining problem is how to support more data disks with less parity disks. Through the cascading Latin scheme introduced in the next section, we can see that if one more parity check disk is used, the amount of data disks supported increases greatly.

IV. Cascading Latin Scheme

1. The union of the basic-systems

In this section, we will construct a new system according to the Latin code based on *L*9 to achieve the aims described in Section II. For convenience, we called the *L*9 based Latin coding

system an L9-basic-system**.** The scheme is described as follows:

Algorithm 2:

Step1 We combine k $L9$-basic-systems into a large system called k-fold system. It is easy to see that the new system can also tolerate double disks failures. It contains $9k$ data disks $\{D_{ij} \mid 1 \le i \le k, 1 \le j \le 9\}$ and $2k$ parity disks $\{P_1, P_2, \ldots, P_k, Q_1, Q_2, \ldots, Q_k\}$ where $\{P_1, P_2, \ldots, P_k\}$ denote the horizontal parity disks, $\{Q_1, Q_2, \ldots, Q_k\}$ denote the Latin parity disks.

Step2 Add a new parity disk named PH, and let

$$PH = \bigoplus_{1 \le i \le k} P_i \qquad (4)$$

Apparently, PH is the sum of all data disks. Now, there are $9k$ data disks and $2k+1$ parity disks in the system. We call this new system PH-system.

Step3 We construct a MDS 3-erasure-correcting system called the $Q3$-system - $\{PH, Q_1, Q_2, \ldots, Q_k, PP_1, PP_2, PP_3\}$, that is, we take $\{PH, Q_1, Q_2, \ldots, Q_k\}$ as data disks and add 3 check disks $\{PP_1, PP_2, PP_3\}$. We name the whole system (including $9k$ data disks and $2k+4$ parity disks) redundant 3-erasure based cascading system.

Note that we have not explained the detail how to calculate PP_1, PP_2 and PP_3, however we know there are code schemes can do this, such as RS code or a generalized EVENODD code.

Step4 Remove $\{PH, P_1, P_2, \ldots, P_k, Q_1, Q_2, \ldots, Q_k\}$ from the redundant 3-erasure based cascading system, the remaining $9k$ data disks and three check disks $\{PP_1, PP_2, PP_3\}$ compose a new system named the 3-erasure based cascading system.

Theorem 2 A 3-erasure based cascading system can tolerate any double disk failure.

Proof There are 3 kinds of double disk failures:

Case 1 The two failed disks are all parity disks.

This case is trivial. We can simply re-encode the failed check disks.

Case 2 A data disk D_{ij} and a check disk PP_j fails.

Because of the failure of D_i, we can not compute PH and Q_i directly. However ,the $Q3$-system is a 3-erasure-correcting system, and we can calculate all other Qs, so we can reconstruct PH, Q_i and PP_j, and then D_i can be reconstructed using PH.

Case 3 Two data disks fail.

Similarly, PH and at most 2 Qs can not be recomputed, we can reconstruct them using the $Q3$-system. However, we can not reconstruct the two failed data disks through PH now, let us consider 2 cases:

Case 3.1 The two failed data disks D_{ij} and D_{ik} belong to the same $L9$-basic-system L_i

Then P_i is the only horizontal parity disk that can not be reconstructed, we can reconstruct it using single-erasure-correcting system $\{PH, P_1, P_2, \ldots, P_k\}$, and then the two failed data disks can be reconstructed though L_j (a 2-erasure-correcting system).

Case 3.2 The two failed data disks D_{ik} and D_{ja} belong to two different $L9$-basic_systems L_i and L_j.

Then we can not compute P_i, P_j, Q_i, Q_j and PH directly. Like Case 2, we can reconstruct Q_i, Q_j and PH using the $Q3$-system, and then reconstruct P_i, P_j and failed data disks in L_i, L_j.

2. 3-erasure-correct scheme

Which 3-erasure-correcting scheme is the best for constructing the $Q3$-system? RS codes is a choice. However, the cascading scheme will not be superior to other 2-erasure-correcting codes in

computational complexity and coding cycle - extensibility conflict of RS code. We can use other 3-erasure-correcting codes, such as STAR code[18],Weaver code[19] or Hover code[20] to overcome the high complexity problem. However, most of them have prime limitation, which may cause serious imbalance between the data disks and the check disks.

Considering the special status of PH, we can design a better scheme:

Algorithm 3:

Step 1 Same as Constructing Algorithm 2.

Step 2 Same as Constructing Algorithm 2.

Step 3 We regard $\{Q_1, Q_2, \ldots, Q_k\}$ as k data disks, and add two parity disks PP_1, PP_2 to construct a $L9$-Basic-System called the $Q2$-system - $\{Q_1, Q_2, \ldots, Q_k, PP_1, PP_2\}$. The whole system contains $9k$ data disks and $2k+3$ parity disks. We call it the *redundant cascading system*.

Step 4 Delete the $2k$ parity disks $\{P_1, P_2, \ldots, P_k, Q_1, Q_2, \ldots, Q_k\}$ from the redundant cascading system, then we get a new system with $9k$ data disks and 3 parity disks $\{PH, PP_1, PP_2\}$, we name it the two-level cascading Latin system. Fig. 3 gives illustration.

Note that k must ≤ 9, otherwise the $Q2$-system is not 2-erasure-correcting.

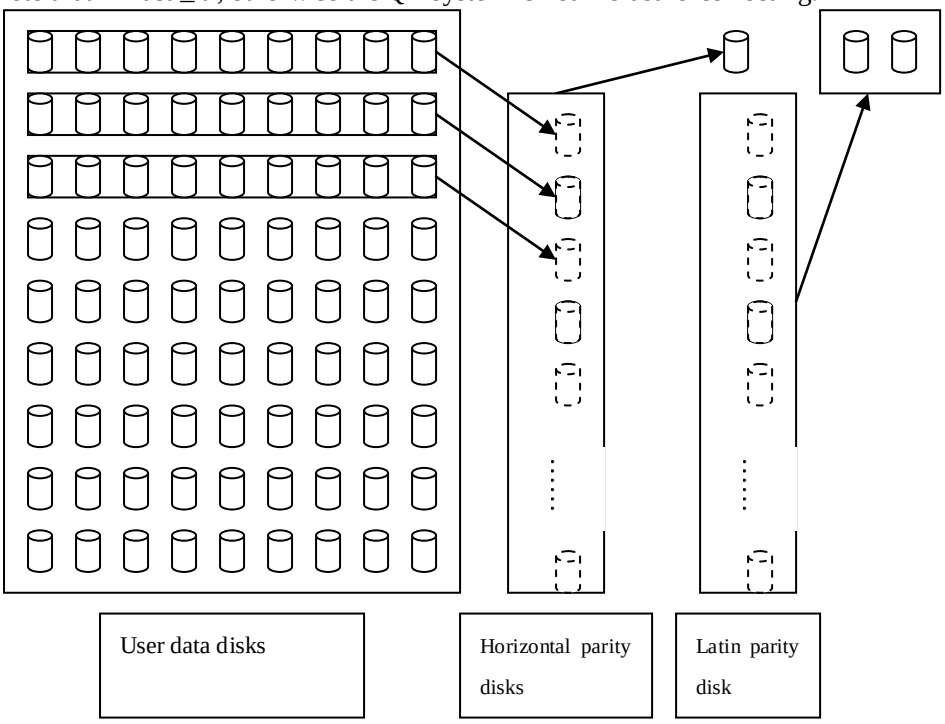

Fig. 3 Illustration of two-level cascading system.

Theorem 3 If $k \leq 9$, the two-level cascading Latin system can tolerate any double disk failure.

Proof There are 4 kinds of double disk failures:

Case 1 The two failed disks both belong to $\{PH, PP_1, PP_2\}$.

We can simply re-compute the two failed parity disks.

Case 2 *PH* and a data disk D_{ij} (belonging to L_j) fail.

Q_i is the only un-reckonable Latin parity disk, we can reconstruct it through the $Q2$-system, then the failed data disk can be reconstructed using L_i, and then PH is recomputed.

Case 3 One failed disk belongs to $\{PP_1, PP_2\}$, without the loss of generality, suppose PP_1 fails. Another failed disk is a data disk D_{ij}.

Firstly, we reconstruct D_{ij} using PH, and then recompute PP_1.

Case 4 Two data disks D_{ik} and D_{ja} fail.

All Latin parity disks can be recomputed except Q_i and Q_j, so we can reconstruct Q_i and Q_j using the Q2-system, and then:

Case 4.1 $i=j$, then P_i can be reconstructed using PH and other horizontal parity disks, and then the two failed data disks are reconstructed in L_i.

Case 4.2 $i\neq j$, then we can reconstruct the two failed disks using Q_i and Q_j.

3. Analysis of two-level cascading Latin systems

Coding cycle: Cascading Latin scheme has a constant coding cycle of 8. It is universal and very easy to be used in large storage system.

Redundancy Rate: 3/n.

Coding length: The number of data disks n should be less than 9k, while k is the number of the L9-basic-systems.

Extensibility: Within 9k data disks, the system is very easy to be extended.

The scheme described above achieves universal property, only at the cost of little extra redundancy, and keeps low encoding/decoding complexity. But this scheme has strong restriction on coding length. The maximum number of data disks is 81, which is much less than that of EVENODD code. If we use L17 as basic-system, the whole system can support up to 17×17=289 data disks.

4. Multi-level cascading latin construction

In step 3 of Algorithm 3, a L9-basic-system is used to tolerate double failures in the Q2-system. However, the L9-basic-system limits the number of data disks that k should be less than 9. We can use a two-level cascading system rather than a L9-Basic-System. It is obvious that the new system can also tolerate any double disk failure, while the system can support up to 9×9 ×9=729 data disks. To achieve this, another more parity check disk should be added to the system, and we should take totally 4 parity disks. Using the Latin scheme in this manner is the reason why we named the scheme “Cascading Latin scheme”.

5. The analysis of multi-level cascading latin system

In a cascading Latin system, the coding cycle keeps a constant number. However, while seeking for the arbitrary system size, the parity disk overhead increases to $[(\log_c n)+1]/n$, c=9 for L9 based cascading Latin scheme. That is to say, we need $\log_c n+1$ check disks to provide 2-erasure-correcting guarantee for n data disks.

Compared with RS code, cascading Latin scheme needs much less XOR operations to do encoding/decoding, the former needs $O(n^3)$ XORs [4], the latter needs only $O(n^2)$ XORs, which is comparative with EVENODD.

The space efficiency is also a very important problem, and it is directly related to the coding cycle. For all other schemes, the coding cycle keeps changing with the number of disks. The best record belongs to the RS code, is about $O(n\lg n)$. To cascading Latin, it needs only $O(n)$ memory.

At last, another characteristic of cascading Latin scheme is its unlimited extensibility. System extension is usually a nightmare for other XOR based codes. However, for the cascading Latin scheme, the only thing needed to do is adding zeroed new disks and doing a bit of XORs.

The update penalty of the cascading Latin scheme is not optimal. To update one data disk, we must change the contents of all the check disks. For two-level cascading Latin scheme, the update

penalty is 3, while the optimal value is 2. However, this weakness may cause sharp performance degradation only for the systems with high load, and most array accesses are small writes. For the system where a few of the accesses are small writes, the cascading Latin scheme will perform as good as other 2-erasure-correcting codes with optimal update penalty.

In a large system, the only alternative of cascading Latin codes is the RS codes. Which scheme is more attractable is due to whether the large amount disk I/Os or the large amount memory and CPU resource used in encoding is the most serious bottleneck of the whole system.

V. Conclusion

From the viewpoint of a system developer, this paper has analyzed the problems of the known XOR based 2-erasure-correcting codes. With a new scheme based on column-Hamiltonian Latin squares - cascading Latin coding scheme, we provide the ability to convert the amount of encode/decode calculation or the inconvenient various coding cycle to a few more redundant parity check disks. With this ability, system developer can decide freely which property is the most important in their system.

Theoretical and practical studies on encoding/decoding algorithms for cascading Latin codes are the most important future works. Generalizing this scheme to multiple erasure correcting is also worth serious study.

References

[1] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software: Practice and Experience*, 27(1999)9, 995-1012.

[2] J S. Plank and Lihao Xu. Optimizing Cauchy Reed-Solomon Codes for Fault-Tolerant Network Storage Applications. In Proceedings of the 5th IEEE International Symposium on Network Computing and Applications (IEEE NCA,06), Cambridge, MA, July 2006,173-180.

[3] Shu Lin and Daniel J. Costello. Error Control Coding Second Edition. ISBN-10: 0130426725, Prentice Hall,2004-06-07 ,156-179

[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 Computing*, 44(1995)2,192– 202.

[5] M. Blaum, J. Brady, J. Bruck, J. Menon, and A. Vardy.The EVENODD code and its generalization.High Performance Mass Storage and Parallel I/, JohnWiley& Sons, INC., 2002, 187-208,.

[6] M. Blaum, J. Bruck, and A. Vardy. MDS array codes with independent parity symbols. *IEEE Transactions on Information Theory*, 42(1996)2, 529-542.

[7] M. Blaum, R. M. Roth. New array codes for multiple phased burst correction. *IEEE Transactions on Information Theory*, 39(1993)1, 66-77.

[8] L. Xu and J. Bruck. X-Code: MDS array codes with optimal encoding. *IEEE Transactions on Information Theory*, 45(1999)1, 272-276.

[9] L. Xu, V. Bohossian, J. Bruck, and D. Wagner. Low density MDS codes and factors of complete graphs.

IEEE Transactions on Information Theory, 45(1999)1, 1817-1826.

[10] Wang Gang, Dong Sha-Sha, Liu Xiao-guang, Lin Sheng and Liu Jing . Construct two-erasure data layout using P1F (利用图的完全 1-因子分解构造双容错数据布局). *ACTA ELECTRONICA SINICA*,34(2006)12A, 2447-2450.

[11] Wang Gang, Lin Sheng, Liu Xiaoguang, Xie Guangjun and Liu Jing.Combinatorial constructions of multi-erasure-correcting codes with independent parity symbols for storage systems. IEEE PRDC 2007, Melbourne, Victoria, Australia, Dec, 2007, 61-68.

[12] Wang Gang, Xiaoguang Liu, Sheng Lin, Guangjun Xie and Jing Liu. Constructing double- and tripe-erasure-correcting codes with high availability using mirroring and parity approaches. icpads, vol. 1, 13th International Conference on Parallel and Distributed Systems - Volume 1 (ICPADS'07), 2007,1-8.

[13] Wang Gang, Lin Sheng, Liu Xiaoguang, Xie Guangjun and Liu Jing.A generalization of RDP code via combinatorial method. Seventh IEEE International Symposium on Network Computing and Applications, 2008 ,93 – 100.

[14] J. S. Plank.The RAID-6 Liberation Codes. 6th USENIX Conference on File and Storage Technologies, San Francisco, 2008, 97–110.

[15] Lisa Hellerstein, Garth A. Gibson, Richard M. Karp, Randy H. Katz, and David A. Patterson. Coding techniques for handling failures in large disk arrays. *Algorithmica*, 12(1994)2/3,182-208.

[16] I. M. Wanless. Perfect factorizations of bipartite graphs and latin squares without proper subrectangles. *The Electronic Journal of Combinatorics*,6(1999)1,R9.

[17] Charles J. Colbourn and Jeffrey H. Dinitz . Handbook of Combinatorial Designs . 2nd ed. (Discrete Mathematics and Its Applications) " , Chapman and Hall/CRC; 2 edition (November 2, 2006),135-151.

[18] Cheng Huang and Lihao Xu. STAR: An efficient coding scheme for correcting triple storage node failures. 4th USENIX Conference on File and Storage Technologies, San Francisco, 2005, 197-210.

[19] J. L. Hafner. WEAVER codes: highly fault tolerant erasure codes for storage systems. In Proceedings of the 4th USENIX Conference on File and Storage Technologies, San Francisco, Dec. 2005, 211-224.

[20] J. L. Hafner. Hover erasure codes for disk arrays. International Conference on Dependable Systems and Networks, Philadelphia, PA, USA, Jun. 2006, 217-226.