ERASURE RESILIENT CODES IN PEER-TO-PEER STORAGE CLOUD

Jin Li (jinl@microsoft.com)
Microsoft Research, Communication and Collaboration Systems, One Microsoft Way, Bld. 113, Redmond, WA 98052

Qiang Huang (qhuang@princeton.edu)
Department of Electrical Engineering, Princeton University, Princeton, NJ 08540.

ABSTRACT

In this paper, we investigate the use of erasure resilient code (ERC) in a peer-to-peer storage cloud. We compare the random linear code (RLC, network coding) and the Reed-Solomon (RS) code, and study the proper ERC parameters in term of data reliability, computational efficiency and security concerns. We conclude that the use of ERC in P2P storage may greatly improve data reliability and reduce backup server cost. Because of the lack of efficient homomorphic hashing, RLC can not be implemented efficiently and yet be effective against the malicious attack. As a result, we believe that the RS code is most suited for use in the P2P storage cloud.

1. INTRODUCTION

Peer-to-peer (P2P) applications, such as Napster, Kazaa, BitTorrent and Skype, have witnessed tremendous success among the end users. A unique characteristic of the P2P application is that the peers bring with them serving capacity when they join the network. Therefore, as more peers participate in the system and the demand of a peer-to-peer system grows, the capacity of the system grows too. Moreover, the majority of the capacity is paid by the peers. This is in sharp contrast to the traditional client-server system, where the server capacity is fixed and paid for by the provider. As a result, the P2P application is cheap to build and superb in scalability.

In this paper, we are particularly interested in P2P storage cloud. In such P2P applications, the peer contributes not only the bandwidth but also the hard drive space to serve the other peers. The collective storage space contributed by the peers form a P2P storage cloud. Data may then be stored into and retrieved from the cloud. P2P storage cloud can be used for a number of applications. One is the distributed backup. The peer may backup its own data into the P2P cloud. When the peer fails, the data may then be restored from the cloud. Another P2P application is the distributed data access. Because the retrieving client may recover the data from multiple data holding peers, the P2P data retrieval may have higher throughput compared with retrieving data from a single source. The other interesting application is on-demand movie viewing [7], in which multiple servers may collaborate with the P2P cloud to serve movies on-demand, as shown in Figure 1. The server may seed the P2P cloud with movies. Then when the peer is viewing the movie, it may stream the movie from both the P2P cloud and the server, thus reduce the server load, reduce traffic on the network backbone and improve the streaming movie quality.

Note that there are extensive research works on building mass reliable storage device in a server cluster, e.g., [2]. However, a P2P storage cloud differs from a server cluster (also called a computer farm or ranch) in three important aspects. First, the peers in a P2P

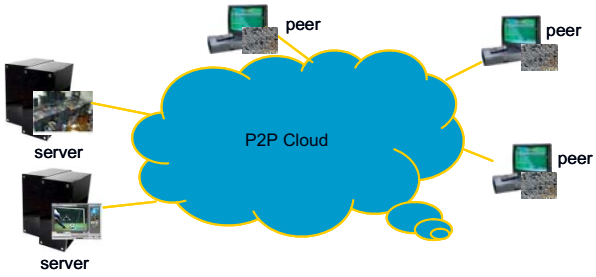

Figure 1 P2P cloud and its role in on-demand movie streaming.

application are distributed over a wide area network (WAN) with large geographical diversity, while the server cluster is formed by a group of networked servers in one location with fast backbone connection. Second, all servers in a server cluster stays in a trustworthy environment, firewall and security mechanism only needs to be implemented at the boundary of the server cluster. This is not the case in a P2P network, where there may be malicious peers who are interested to explore the weakness of the protocol. Third, the reliability of the peer and its network connection is several orders of magnitude lower than that of the server. Because the peer is usually an ordinary consumer computer that is running the P2P software and contributing its spare hard drive space and idle bandwidth resource, the computer or the P2P application might be shut down by the user from time to time for a variety of reasons, e.g., the need to upgrade and patch the software, the need to install/update hardware, virus attack, etc.. The user may also launch new applications that compete in CPU and bandwidth resource with the P2P application. The computer may also connect to the network through an unreliable ISP or unreliable WiFi link. Building reliable and high performance data storage cloud in an unreliable, untrustworthy P2P network thus poses unique challenge.

A basic tool for achieving data reliably in unreliable P2P cloud is high rate erasure resilient code (ERC). The important design questions then become: How many copies of the data should be stored? In what form should the data be stored? What is the appropriate parameter for ERC?

In the rest of the papers, we will investigate the use of ERC in the P2P storage cloud. Two important ERCs: the Reed-Solomon (RS) code and random linear code (RLC) are briefly reviewed in Section 2. We study the reliability of P2P storage with and without ERC, the difference between the RS code and RLC, and the appropriate ERC parameters in Section 3. The security concern of P2P storage and its implication in the selection of ERCs is investigated in Section 4. We study the problem of P2P storage cloud with backup server support in Section 5. The mathematical analysis we used in the paper is shown in Appendix I.

2. ERC: REED-SOLOMON (RS) AND RANDOM LINEAR CODE (RLC)

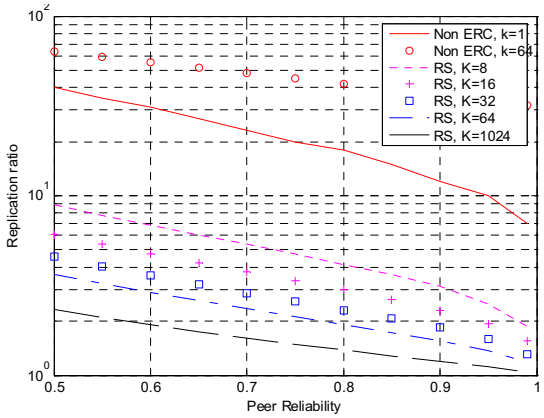

Figure 2 Peer reliability and desired replication ratio.

ERCs are block error correction codes. In general, ERC segments an original message into k original fragments $\{x_i\}$, $i = 0,\cdots, k-1$, each of which is a vector over the Galois Field GF(q), where q is the order of the field. Say we are encoding a message that is 64KB long, if we use GF(2^{16}) and k=16, each fragment will be 4KB, and will consist of 2K word, with each being an element of GF(2^{16}). An ERC coded fragment is formed by operation:

$$c_j = \mathbf{G}_i \cdot [x_0 \quad x_1 \quad \cdots \quad x_{k-1}]^t, \tag{1}$$

where c_j is a coded fragment, $\mathbf{G_i}$ is a k-dimensional generator vector, and equ (1) is a matrix multiplication, all on GF(q). At the time of decoding, the peer collects m coded fragments, where m is a number equal to or slightly larger than k, and attempts to decode the k original fragments. This is equivalent to solve the equation:

$$\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{bmatrix} = \begin{bmatrix} \mathbf{G_0} \\ \mathbf{G}_1 \\ \vdots \\ \mathbf{G}_{k-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{k-1} \end{bmatrix}, \tag{2}$$

If the matrix formed by the generator vectors has a full rank k, the original messages can be recovered.

There are many existing ERCs. Two types of ERCs that are particularly interesting is Reed-Solomon (RS) code[1] and random linear code (RLC, also called network coding)[3]. The RS code uses structured generator vector, e.g., based on the Vandermonde matrix:

$$\mathbf{G}_i = \left[i^0 \quad i^1 \quad \cdots \quad i^{k-1} \right], \tag{3}$$

or based on the Cauchy matrix:

$$\mathbf{G}_i = \begin{cases} [0 \quad \cdots \quad 0 \quad 1 \quad 0 \quad \cdots \quad 0] & i < k, \text{ only } i\text{th position is 1} \\ \left[\frac{1}{\text{i+0}} \quad \frac{1}{\text{i+1}} \quad \cdots \quad \frac{1}{\text{i+(k-1)}} \right] & i \geq k \end{cases}, \tag{4}$$

Both RS codes above are maximum distance separable (MDS) codes. As a result, any k distinctive coded fragments will be able to decode the original message. Because RS codes are structured, RS coded fragment can be easily identified and managed by the index i of the generator vector. This eases the detection of duplicated RS codes.

In comparison, RLC (network coding) uses a random linear vector for each code. The chief advantage of RLC is that its coded fragment can be further combined by the peer to generate new code. Let there be two coded fragment generated via vector:

$$c_0 = [\alpha_0 \quad \alpha_1 \quad \cdots \quad \alpha_{k-1}] \cdot [x_0 \quad x_1 \quad \cdots \quad x_{k-1}]^t, \text{ and} \tag{5}$$
$$c_1 = [\beta_0 \quad \beta_1 \quad \cdots \quad \beta_{k-1}] \cdot [x_0 \quad x_1 \quad \cdots \quad x_{k-1}]^t$$

where $\{\alpha_i\}$ and $\{\beta_i\}$ are generator vectors with their elements randomly generated. An intermediate receiving peer may combine the two coded fragment into a new coded fragment:

$$c_2 = \xi \cdot c_0 + \zeta \cdot c_1 = [\chi_0 \quad \chi_1 \quad \cdots \quad \chi_{k-1}] \cdot [x_0 \quad x_1 \quad \cdots \quad x_{k-1}]^t, \tag{6}$$
with $\chi_i = \xi \cdot \alpha_i + \zeta \cdot \beta_i$

where ξ and ζ are two random non-zero elements in GF(p). Obviously, any linear combination of RLC codes is still a valid RLC code. RLC store the generator vector together with the coded fragment, i.e., attaching $\{\alpha_i\}$ to c_0, $\{\beta_i\}$ to c_1, and $\{\chi_i\}$ to c_2. At the time of decoding, a generator matrix is formed with the generator vectors attached to the coded fragments. The RLC decoder will examine the generator matrix and determine if the associate coded fragments can decode the original coded message.

Table 1. Desired replication ratio to achieve 12 nines reliability.

ERC Methods	Reliability p=0.5		p=0.9	
	r	# of peers	r	# of peers
Non ERC, k=1	40	40	12	12
Non ERC, k=64	63.34	4054	36.58	2341
RS, k=8	8.88	71	3.13	25
RS, k=16	6.13	98	2.31	37
RS, k=32	4.56	146	1.84	59
RS, k=64	3.64	233	1.56	100
RS, k=1024	2.34	2391	1.20	1226

3. RELIABILITY IN P2P STORAGE CLOUD

In our first analysis, we show how ERC improves the reliability of the P2P cloud. Let there be r copies of messages in the P2P cloud. Let the reliability of each peer be p (the peer is available with probability p). We show in Figure 2 (with number listed in Table 1) the desired replication ratio r if we want the data stored in the P2P cloud to achieve a failure rate smaller than 10^{-12} (i.e., 12 nines reliability). In Figure 2, the horizontal axis is the peer availability, and the vertical axis is the desired replication ratio (in log scale). The mathematics behind the analysis can be found in Appendix I. We compare seven approaches:

a) Non erasure coding, the entire message is replicated to r peers.
b) Non erasure coding, split message into k=64 fragments, then replicate fragments to $r \cdot k$ peers.
c) RS code, split into k=8 fragments, and distribute fragments to $r \cdot k$ peers. We remember RS index so that there are no duplicated RS codes in the P2P cloud.

d), e), f) g) Same as c), except k=16, 32, 64, and 1024.

To achieve the same reliability, the use of ERC reduces the required replication ratio by an order of magnitude. For example, with peer reliability of p=0.5, we may achieve 12 nines reliability by either storing the data into 40 peers without using ERC, or storing a coded fragment 1/1024 the size of the message into 2391 peers, leading to an actual replication ratio r=2.335. The latter approach greatly reduces the amount of the hard drive that the peer needs to contribute to the P2P cloud in exchange for reliable data backup.

Note that the simple act of fragmenting the message without using ERC reduces reliability. Shown in scheme b), if messages are split into 64 fragments and distributed to the peers without ERC, we will need replication ratio r=63.34. This represents a 58% and 1640% increase in the replication ratio compared with non ERC distribution of whole message and ERC distribution of the message with same fragmenting parameter (k=64).

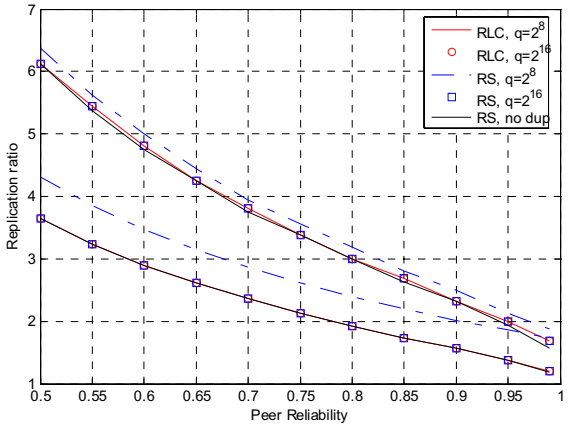

Figure 3 RS code vs RLC code (except RS code with $q=2^8$, the required replication ratio of the rest codes are very close.)

We also observe that splitting a message into a large number of fragments may further reduce the required replication ratio. With peer reliability of p=0.5, the required replication ratio decreases from r=8.875 to r=2.34 as we increase the fragment size from k=8 to k=1024. The replication ratio decreases by a factor of 3.8:1. With peer reliability increases to p=0.9, splitting the messages into 1024 instead of 8 fragments may still reduce the required replication ratio by a factor of 2.6:1. However, this comes at a cost of flexible information storage and retrieval. The large fragment size requires the coded fragment be stored into and retrieved from thousands of peers (2391 for p=0.5, 1226 for p=0.9). This is impractical. Even for P2P backup, where data is not expected to be updated frequently, the use of fragment size k=1024 means that every time a file is modified, the user needs to contact thousands of peers to update the data. The increase in the administer traffic and the response time makes such system impractical. In the authors' opinion, ERC fragmentation with size k=16 to k=64 is more practical in real-world applications.

Please note that storing multiple fragments into a peer node does not improve the reliability. For example, if we split the message into 64 fragments, and store 4 fragments in each peer node, the reliability will be the same as splitting the message into 16 fragments with each peer storing one fragment. Allowing more fragments to be stored in a peer do help to balance the load in the P2P cloud, as peers with more storage space may store more fragments, and peers with less storage space may store fewer fragments.

In Figure 3, we compare the RS code with RLC code. The detail mathematical analysis can be found again in Appendix I. We still calculate the desired replication ratio in order to achieve 12 nines reliability with varying peer reliability. The following five schemes are compared:

a) RLC code, split into k=16 or 64 fragments, and distribute the fragments into $r \cdot k$ peers. We use GF(2^8).
b) Same as a), except GF(2^{16}) is used.
c) Same as a), but use RS code. We do not detect duplicate RS code.
d) Same as c) except GF(2^{16}) is used.
e) Same as c) and no duplicate RS code in the system.

We observe except for RS code on GF(2^8), the rest of ERCs perform almost the same. The experiment shows that RLC code and RS code achieve almost the same reliability. It also shows that if we use RS code in a P2P storage cloud, we should either use a large Galois Field, e.g., GF(2^{16}) or make sure that no two nodes in the P2P cloud hold the same RS code. The latter approach can be done in RS code as the code can be indexed with a single integer value i, which is the index of the generator matrix. This does require that the source peer remembers the RS code assigned to the peer node, which incurs additional administrative work on the source peer.

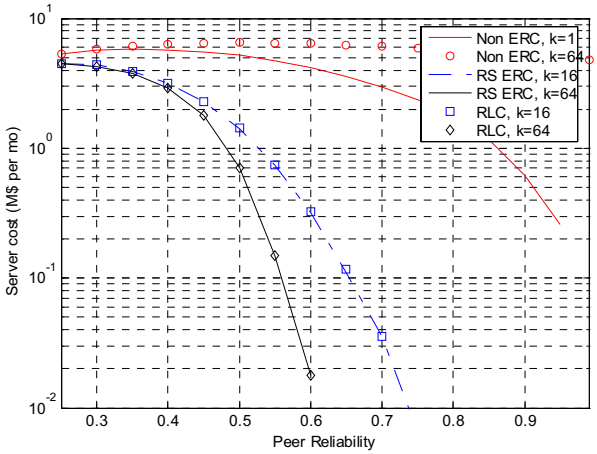

Figure 4 P2P cloud backed by server: server cost.

4. SECURITY IN P2P STORAGE CLOUD

Without proper security measure, the P2P storage cloud is susceptible to the denial of service (DOS) attack. An example of such attack can be found in [4], in which a malicious peer pollutes the content in the P2P network by injecting garbage into the network. DOS attack in P2P cloud can be especially malicious as a single bit error in a single ERC coded message may propagate through ERC decoding and pollute all data stored in the P2P cloud. For non ERC and RS coded fragment, we may use well known hash (e.g., SHA-1) and signature (e.g., RSA) approach to guard the integrity of the content. The user may simply publish his public key, and sign every coded fragments originated from the user with the user's private key. Whenever a coded fragment is received by a certain peer, the signature is verified by the recipient. If the signature does not match, the recipient will throw away the corrupted fragment and report the sender as malicious to a central authority.

The guard against DOS attack for RLC coded fragment is trickier. The idea is to use homomorphic hashing and homomorphic signature scheme[5][6]. Homomorphic hash function is linear and is computationally infeasible to find collisions. The linear property will allow us to compute the signature of a linear combination of the input messages given the original signatures of the input, and allow us to check whether a message has been altered, hence allow the mixing of the coded fragments in equ (6). However, homomorphic signature scheme works only on large Galois Field, e.g., with the order of the field q being a prime about 170bits long[6]. This greatly increases computational complexity in the RLC encoding and decoding. Moreover, the computation of the homomorphic hashing, the calculation and verification of the signature is also much computational intensive. Overall, to ensure secured data distribution, RLC is currently two-orders of magnitude [6] or three-orders of magnitude ([5], throughput 160kbps) slower than RS code (throughput 80Mbps). The lack of more efficient homomorphic hashing may hamper the adoption of RLC in real P2P applications.

5. P2P STORAGE CLOUD WITH SERVER BACKUP

In commercial P2P deployment, it is not uncommon for the P2P storage cloud to be backed by a reliable server. One example is the

on demand movie viewing application shown in Figure 1. In this case, the server holds a complete copy of the movie. The P2P cloud is used as a supplement to aide the server in delivering the movie. With the existence of a reliable backup server, the data in the P2P cloud is always reliable. However, the server does incur cost (mainly in term of bandwidth) to serve the peer if the data can not be found in the P2P cloud. We assume that the P2P cloud consists of one million users. And each peer when available, stream a high definition movie at 7Mbps (WMV HD bitrate). We assume that all messages in the P2P cloud will be distributed with a replication ratio r=2. Whenever a message can not be decoded because there are insufficient number of coded fragments in the P2P cloud, the server will step in and serve the extra coded messages. We calculate the probability distribution of the server load (detailed in Appendix I) and calculate the bandwidth cost of the server by the 95th percentile rule[1], assuming the billing rate is 50$ per month per Mbps. We show the server cost (in million dollars per month, in the vertical axis) at different peer reliability rate (the horizontal axis) in Figure 4.

We compare six approaches:

a) Non ERC, store entire messages to r (2) peers.
b) Non ERC, split message into k=64 fragments, then store the fragments to $r \cdot k$ (128) peers.
c) RS code, split into k=16 fragments, and store the fragments into $r \cdot k$ (32) peers.
d) Same as c), except k=64.
e), f) Same as c) and d), but use RLC.

We observe that the use of ERC greatly reduces the bandwidth cost of the server. The cost saving is minor when the peer reliability is poor ($r \cdot p$<0.5), as there are insufficient numbers of coded fragments in the P2P cloud, and the peers have to go to the server for the missing fragments anyway. However, the cost saving becomes significant when the peer reliability increases. The use of ERC makes sure that the P2P cloud does not suffer from the coupon collector's problem, and the server is only consulted occasionally as a last resource.

We also observe from Figure 4 that the performance of the RLC and RS code is similar. Again, splitting messages into more fragments helps to further reduce the server cost at the expense of more complicated system design and content retrieval protocol.

APPENDIX I. MATH OF RELIABILITY ANALYSIS

Without using ERC, if the original messages is stored onto r peers, the reliability of the message is p^r.

Analyzing the reliability of ERC contains two parts. First, assume that there are l peers in the cluster, the probability $p_a(m,l)$ that there are exactly m peers available can be calculated via binomial distribution:

$$p_a(m,\ l) = \binom{l}{m} p^m (1-p)^{l-m} \qquad (7)$$

Second, we calculate the probability $p_r(j,m)$ that the generator matrix is of rank j after receiving m coded fragments. For RS code with no duplicate fragments (recall RS code may be identified with index i, managing the code and avoid code duplication is relative easier), then m coded fragments result in a generator matrix with rank min(m,k). If there are possibly duplicated RS codes, then the probability p_r (j,m) that the generator matrix is of rank j after receiving m code is:

$$p_r(1,\ 1) = 1,$$
$$p_r(j,\ 1) = 0,\ j \neq 1$$
$$p_r(j,\ \text{m}) = \begin{cases} p_r(j-1,\ \text{m-1}) \cdot \dfrac{q-j+1}{q} + p_r(j,\ \text{m-1}) \cdot \dfrac{j}{q} & j < k \\ p_r(k-1,\ \text{m-1}) \cdot \dfrac{q-k+1}{q} + p_r(k,\ \text{m-1}) & j = k \end{cases} \qquad (8)$$

Note that splitting a message into k fragments and storing the fragments into the P2P store without ERC can be considered as using a RS code with q=k. Similarly, for RLC codes, we can recursively calculate the probability p_r (j,m) as:

$$p_r(j,\ \text{m}) = \begin{cases} p_r(j-1,\ \text{m-1}) \cdot \left[1 - \left(\dfrac{1}{q}\right)^{k-j+1}\right] + p_r(j,\ \text{m-1}) \cdot \left(\dfrac{1}{q}\right)^{k-j} & j < k \\ p_r(k-1,\ \text{m-1}) \cdot \dfrac{q-1}{q} + p_r(k,\ \text{m-1}) & j = k \end{cases} \qquad (9)$$

Combing (7)-(9), we can calculate the probability $p_s(j)$ which records the probability that there are j linearly independent coded fragment in the P2P network as:

$$p_s(j) = \begin{cases} p_a(0,l) & j = 0 \\ \sum_{m=1}^{l} p_a(m,l) \cdot p_r(j,m) & j > 0 \end{cases}. \qquad (10)$$

The probability of data availability is simple the probability p_s (k). To calculate a desired replication ratio for a certain failure rate f, we increase the replication ratio r untill the available probability is larger than 1-f.

In the analysis of the P2P cloud backed by the server, we use the p_s (j) to find the number of fragments that needs to be served by the server. The server load vector is:

$$\left[p_{store}(k) \quad p_{store}(k-1) \quad \cdots \quad p_{store}(0)\right], \qquad (11)$$

where the ith element of load vector is the probability that the server needs to serve i coded fragments to the peer. Convolution of the server load vector will lead to the probability distribution of the server load, which can then be used to calculate the server cost.

[1] 95th percentile is a billing mode used by most ISPs. To calculate the billing traffic rate, the ISP measures the load of the server every five minutes, 24 hours a day, every day of the month. The collections of five minute load of the server (called samples) are sorted. The ISP ignores the highest 5% of the load, and the value of the next highest load sample becomes the 95th percentile value at which you will be billed.

11. REFERENCES

[1] B. W. Stephen and V. K. Bhargava, "Reed-Solomon codes and their applications", Wiley Press, 1999.

[2] Q. Xin, et. al., "Reliability mechanisms for very large storage systems", 20th IEEE/11th NASA Goddard Conf. on Mass Storage Systems and Technologies (MSS'03), pp. 146-156.

[3] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," Allerton Conference on Communication, Control, and Computing, Monticello, IL, October 2003. Invited paper.

[4] "HBO Attacking BitTorrent", http://radar.oreilly.com/archives/2005/10/hbo_attacking_bittorrent.html

[5] C. Gkanstsidis, P. Rodriquez, "Cooperative security for network coding file distribution", Tech. Rep. MSR-TR-2004-137, Microsoft Research, 2004.

[6] D. Charles, K. Jain, K. Lauter, "Signatures for network coding", under preparation, 2005.

[7] J. Li, "PeerStreaming: A practical receiver-driven peer-to-peer media streaming system", MSR-TR-2004-101, Sept. 2004.