Maintenance-Efficient Erasure Coding for Distributed Archival Storage

Cuong Pham[†] Feng Zhang[‡] Duc A. Tran[†]
[†] Department of Computer Science, University of Massachusetts, Boston
[‡] Distributed System Infrastructure Group, EMC Corporation, USA
Email:{cpham, duc}@cs.umb.edu, feng.zhang@emc.com

***Abstract*—Maintenance is an important issue in distributed storage systems. In many cases, such as in archival storage where data access is infrequent, the workload due to maintaining a system is dominant and much heavier than that to answering retrieval queries. Erasure coding, widely used in many distributed storage systems for its high reliability, is efficient for access but not so efficient for maintenance. In this paper, we investigate an extension of this method with the purpose of improving maintenance efficiency and provide an analysis on the tradeoff of this extension.**

***Index Terms*—Erasure Coding, Distributed Storage, Maintenance Efficiency**

I. Introduction

In recent years, with the rapid growth of the Internet and users' generated content, reliability is an important issue for any distributed data storage system that wants to provide 24-hour service to its users (e.g., online banking, Google search, Facebook access, to name a few). To have a high degree of availability, the data should be replicated across many servers, so that when some servers go down, the alive ones can still provide the requested data. Erasure coding has been suggested as a more reliable method of replication than full replication [14], [17]. Erasure coding involves two steps. First, the original data is split into k equi-size blocks. Second, these blocks are used to generate n encoded blocks each being stored at a random server. This way, we can reconstruct the original data by obtaining any k encoded blocks. Therefore, we can tolerate up to $n - k$ server failures.

A problem with erasure coding is the maintenance cost. To recover from the loss of an encoded block, three tasks are required: (1) k encoded blocks need to be retrieved from k servers; (2) the original data needs to be reconstructed from these k blocks; (3) a new encoded block needs to be generated to replace the lost one. In a large-scale distributed storage system with millions of files stored across thousands of servers, failures occur very frequently resulting in many encoded blocks lost. The maintenance cost incurred by performing all the three tasks above can be too expensive in terms of both communication and computation, because the entire file needs to be reconstructed just to replace a single block. This cost is dominant in archival systems where data is accessed infrequently but we need to make sure that the data is always there when needed.

This paper is focused on such archival systems. Specifically, we investigate REC [1] – an extension of traditional erasure coding (TEC) with the purpose to reduce the maintenance cost. Instead of storing a single copy for each encoded block, REC stores multiple copies and distribute these copies on the servers in a way that is convenient to recover from any failure. REC results in the same file availability as TEC but offers much better efficiency when it comes to maintenance. We have implemented REC in a real-world prototype on top of the CitySense network [1]. CitySense is a wireless mesh network deployed in Cambridge, MA to monitor its environment.

The remainder of this paper is structured as follows. Section II discusses related work. Section III describes REC's details. Section IV presents the result of our analysis on the tradeoff of REC in comparison to TEC. Section V describes the implementation of REC on CitySense network. Section VI concludes our paper.

II. Related Work

Erasure coding is widely implemented in distributed storage systems [14]- [17]. It is desirable for its efficient use of storage space given the same data availability [17]. There are different erasure coding techniques such as Reed-Solomon [19], LDPC [20], Tornado [21], LT [22], Raptor [23], to name a few, each having its own class of applications. For instance, in a BitTorrent-like file sharing system, the Tornado code is used to generate encoded blocks without the need to predefine the number of blocks to generate. With Tornado code, the receiver keeps receiving encoded blocks until it can decode to get the original file. Usually the number of encoded blocks required to decode in this case is high. In applications where this number needs to be small, Reed-Solomon is more appropriate.

The maintenance of redundant data blocks is critical in many large-scale reliable distributed storage systems [2]- [10]. In these systems, server failures are inevitable and the lost encoded blocks need to be replaced to maintain the data availability over time. Several techniques have been proposed to make the maintenance more efficient. For example, FARM [9] stored multiple copies of encoded blocks but because of its randomness in choosing where to store these copies, the time to search for the missing blocks might be significant. CFS [8] distributes encoded blocks across a number of servers which are organized in a Chord [25] overlay network. In

[1] REC = redundant erasure coding

contrast, REC is suitable for a server farm instead of an overlay network of servers like Chord. OceanStore [4] replicates the original file blocks instead of encoded blocks. Network Coding (NC) is used in [11] to reduce the bandwidth required for maintenance. The idea is to retrieve fewer encoded blocks than usual but sufficient to reconstruct the original file, by combining encoded blocks at immediate nodes pro-actively. REC does not use NC while requiring less bandwidth to replace the missing blocks.

Total Recall [7] suggests that it is not always a must to repair a lost block and the decision to do this depends on the severity (transient or permanent) of the failure. Total Recall organizes servers in a Chord ring for ease of decentralized management and uses erasure coding for encoding data blocks. A similar work, Carbonite [2] is also about repair policy, but aims at pure replication instead of erasure coding. Unlike these two techniques, all server failures in REC are considered permanent and require immediate repair.

III. Proposed Solution

We consider a large-scale distributed archival storage system of N servers and M files that need to be archived, with following requirements:

- $a \in (0,1)$: a real value representing the server availability, defined as the probability that a server is alive at any given time.
- $A \in (0,1)$: a real value representing the file availability, defined as the probability that a file is accessible at any given time.
- S: a real value representing the storage requirement to store each file (system redundancy factor). Assuming that each file's size is one byte, the storage requirement in the system for each file is S (bytes).

In TEC(k_T, n_T), where k_T and n_T are the number of original data blocks and the number of encoded blocks, respectively, each file B is divided into k blocks of equal size, $\{B_1^T, B_2^T, ..., B_{k_T}^T\}$, which are used to generate n encoded blocks, $\{B_1'^T, B_2'^T, ..., B_{n_T}'^T\}$. Each of these encoded blocks is stored at a random server. Therefore, we have a group of n_T servers to store the n_T encoded blocks for each file. The file availability of this scheme is, according to [11]

$$A_{TEC}(n_T, k_T) = 1 - \sum_{i=0}^{k_T-1} \binom{n_T}{i} a^i (1-a)^{n_T-i} \quad (1)$$

The storage requirement is

$$S_{TEC} = n_T / k_T \quad (2)$$

Based on Eqs. 1 and 2, given A, a and S, we can choose the values for n_T and k_T accordingly. Figure 1(a) shows an example of TEC(k_T,n_T).

We propose REC below as an extension of TEC in order to achieve better efficiency for maintenance. There are three parameters for REC: (1) k_R: the number of original data blocks; (2) n_R: the number of encoded blocks; and (3) r: the number of copies per encoded block, called the block redundancy factor

A. Data Encoding and Storage

Similar to TEC, each file B is divided into k blocks of equal size, $\{B_1^R, B_2^R, , B_{k_R}^R\}$, which are used to generate n encoded blocks, $\{B_1'^R, B_2'^R, , B_{n_R}'^R\}$. A group of n_R random servers is selected to store these encoded blocks. The difference between REC and TEC is that r copies of each encoded block are stored instead of one copy. The storage requirement of REC, therefore, is

$$S_{REC} = r \times n_R / k_R \quad (3)$$

Across the n_R randomly selected servers, the r copies for the encoded blocks are placed in a round-robin fashion as follows: server 1: $\{B_1'^R, B_2'^R, ..., B_r'^R\}$, server 2: $\{B_2'^R, B_3'^R, ..., B_{r+1}'^R\}$, ..., server i: $\{B_i'^R, B_{i+1}'^R, ..., B_{(i+r) \bmod n_R}'^R\}$,..., server n: $\{B_{n \bmod n_R}'^R, B_{(n+1) \bmod n_R}'^R, ..., B_{(n+r) \bmod n_R}'^R\}$.

An example is shown in Figure 1(b), where $r = 2$. There are n_R encoded blocks distributed over the n_R randomly selected servers. The number of copies for each encoded block $r = 2$, thus each server stores two different encoded blocks in round-robin fashion. The first server stores $\{B_1'^R, B_2'^R\}$, the second $\{B_2'^R, B_3'^R\}$, and the last $\{B_{n_R}'^R, B_1'^R\}$.

To retrieve the original file, we need to get k_R different encoded blocks. This can be done by contacting any $m = \lceil \frac{k_R}{r} \rceil$ servers that do not store a common encoded block of the file; for example, the following group of servers:

server 1: $\{B_1'^R, B_2'^R, ..., B_r'^R\}$
server r+1: $\{B_{r+1}'^R, B_{r+2}'^R, ..., B_{2r}'^R\}$
server 2r+1: $\{B_{2r+1}'^R, B_{r+2}'^R, ..., B_{3r}'^R\}$
...
server (m-1)r+1: $\{B_{(m-1)r+1}'^R, B_{(m-1)r+2}'^R, ..., B_{mr}'^R\}$

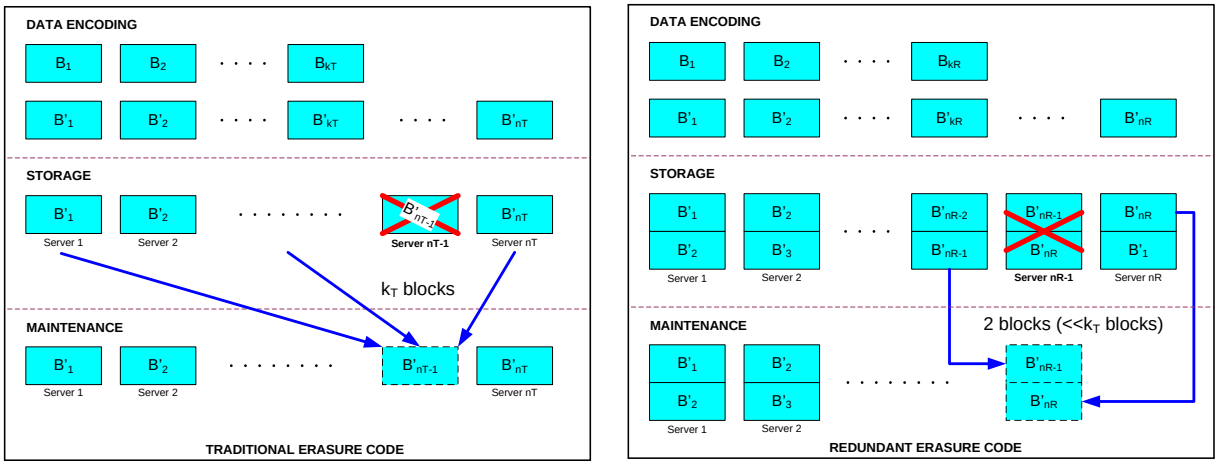
However, it is not always the case that all of these m servers are alive. In the worst case, we have to contact $k_R - r + 1$ servers to reconstruct the file (these servers correspond to the case that the $k_R - r + 1$ servers are consecutive servers). Therefore, the file availability is

$$A_{REC}(n_R, k_R) = 1 - \sum_{i=0}^{k_R-r} \binom{n_R}{i} a^i (1-a)^{n_R-i} \quad (4)$$

Based on Eqs. 3 and 4 and given $A_{REC} = A$, a and $S_{REC} = S$, we can choose the values for n_R and k_R accordingly.

B. Maintenance

When a server fails, all the files that have blocks stored in this server are affected. For each affected file, we need to replace the failed server with an alive server which will store the lost blocks for this file. For example, consider a file B distributed in a group of servers {1, 2, ..., n}. When server i fails, we need to find a server among the servers outside this group to replace this server. We need to obtain the lost blocks $\{B_i'^R, B_{i+1}'^R, ..., B_{(i+r) \bmod n_R}'^R\}$ and store them in the new server.



(a) TEC (b) REC

Fig. 1. Maintenance: REC vs.TEC

The key advantage of REC is that these lost blocks can be obtained easily by contacting only two servers: $(i-1)$ and $(i+1)$. The blocks are just copied from these servers to the new server without the need to perform an expensive 3-step reconstruction of TEC mentioned earlier. Back to the example in Figure 1(b), when server $n_R - 1$ failed, a new server is chosen to be the replacement in the ordered server-list, and this server get the missing $B'^R_{n_R-1}$ and $B'^R_{n_R}$ from server n_R-2 and n_R in the server-list.

IV. Analysis

We analyze REC and TEC for six configurations of server availability ($a \in \{0.8, 0.9\}$) and file availability ($A \in \{0.999, 0.9999, 0.99999\}$). The parameters for REC(r, n_R, k_R, S_R) and TEC(n_R, k_R, S_R) are calculated based on Eqs. 1-2 and Eqs. 3-4, respectively.

A. $a = 0.8$

Table I and Table II show the parameters for REC and TEC that can be used in a real data archival system with server availability $a = 0.8$ to achieve file availability from three-nine to five-nine. For TEC, it is desirable to a have small value for k_T because the maintenance of a single encoded block requires at least k_T other encoded blocks to be retrieved. In Table I, although we can achieve a three-nine file availability with a low system redundancy factor $S_T = 1.484848$, the corresponding number of original data blocks $k_T = 66$ is too high, much higher than the case $k_T = 9$ whose storage requirement increases only by about 34% ($S_T = 2.0$). Therefore, $k_T = 9$, $n_T = 18$ should be the best choice for TEC to achieve three-nine file availability. At a very high file availability of five-nine, the system redundancy factor required for TEC is less than 2.84, which is reasonable for today's most distributed storage systems.

On the other hand, as shown in Table II, to achieve the same file availability of TEC, REC requires more storage. We should choose the values for k_R, n_R, and r so that this cost is minimum. The best choices for (r, k_R, n_R) are (2, 67, 98), (2, 64, 98) and (2, 58, 93) in order to achieve three-nine, four-nine, and five-nine file availability, respectively. The storage required by REC is less than 1.5 times that required by TEC and this gap is getting smaller as we increase the requirement on file availability (three-nine to four-nine to five-nine). The values for n_R and k_R in these choices are large but within a reasonable range (all less than 100), because REC is designed for archival purposes where access is infrequent and maintenance is much more needed.

B. $a = 0.9$

In this study, we assume a more stable network where the server availability is $a = 0.9$. Table III and Table IV show the parameters for REC and TEC to achieve file availability from three-nine to five-nine. In this more stable network, the storage requirement of TEC drops from 2 to 1.625 (18% drop) in the case of three-nine, and, in the case of five-nine, from 2.8333 to 2.5 (11%). The same trend applies to REC, where the storage cost is reduced by 17% (from 2.92 to 2.46) and 18% (from 3.2 to 2.6) in the cases of three nine and five-nine, respectively.

Figure 2 shows the ratio of the system redundancy factor of REC to that of TEC (S_{REC}/S_{TEC}). It is observed that as better file availability is required, REC is getting closer to TEC in terms of storage cost. For example, when $a = 0.9$ and $A = 0.99999$, TEC and REC require almost identical a storage cost.

V. Implementation on CitySense

We have implemented REC on Citysense [1] which is a city-wise wireless sensor network open for research. This network consists of some 100 sensor nodes (indoor and outdoor). Outdoor nodes are mounted on top of street light poles in

TABLE I
TEC PARAMETERS WITH $a = 0.8$

$A = 0.999$		
k_T	n_T	S_T
9	18	2.000000
11	21	1.909091
13	24	1.846154
15	27	1.800000
17	30	1.764706
24	40	1.666667
29	47	1.620690
31	50	1.612903
34	54	1.588235
36	57	1.583333
39	61	1.564103

$A = 0.999$ continued		
k_T	n_T	S_T
41	64	1.560976
44	68	1.545455
47	72	1.531915
49	75	1.530612
52	79	1.519231
55	83	1.509091
57	86	1.508772
60	90	1.500000
63	94	1.492063
66	98	1.484848

$A = 0.9999$		
k_T	n_T	S_T
9	20	2.222222
29	50	1.724138
31	53	1.709677
36	60	1.666667
38	63	1.657895
43	70	1.627907
48	77	1.604167
53	84	1.584906
58	91	1.568966
61	95	1.557377
63	98	1.555556

$A = 0.99999$		
k_T	n_T	S_T
6	17	2.833333
9	22	2.444444
14	30	2.142857
34	60	1.764706
36	63	1.750000
43	73	1.697674
45	76	1.688889
50	83	1.660000
52	86	1.653846
55	90	1.636364
57	93	1.631579

TABLE II
REC PARAMETERS WITH $a = 0.8$

$A = 0.999$			
r	k_R	n_R	S_R
2	67	98	2.925373
3	68	98	4.323529
4	69	98	5.681159
5	13	18	6.923077
6	14	18	7.714286
7	15	18	8.400000
8	16	18	9.000000
9	17	18	9.529412

$A = 0.9999$			
r	k_R	n_R	S_R
2	64	98	3.062500
3	65	98	4.523077
4	64	95	5.937500
5	65	95	7.307692
6	14	20	8.571429
7	15	20	9.333333
8	16	20	10.000000
9	17	20	10.588235

$A = 0.99999$			
r	k_R	n_R	S_R
2	58	93	3.206897
3	59	93	4.728814
4	60	93	6.200000
5	61	93	7.622951
6	60	90	9.000000
7	12	17	9.916667
8	13	17	10.461538
9	14	17	10.928571

TABLE III
TEC PARAMETERS WITH $a = 0.9$

$A = 0.999$		
k_T	n_T	S_T
8	13	1.625000
24	33	1.375000
28	38	1.357143
33	44	1.333333
38	50	1.315789
43	56	1.302326
48	62	1.291667
53	68	1.283019
54	69	1.277778
59	75	1.271186
64	81	1.265625
65	82	1.261538
70	88	1.257143
76	95	1.250000

$A = 0.9999$		
k_T	n_T	S_T
7	13	1.857143
10	17	1.700000
13	21	1.615385
24	35	1.458333
33	46	1.393939
37	51	1.378378
42	57	1.357143
47	63	1.340426
51	68	1.333333
56	74	1.321429
61	80	1.311475
67	87	1.298507
72	93	1.291667
77	99	1.285714

$A = 0.99999$		
k_T	n_T	S_T
4	10	2.500000
18	29	1.611111
25	38	1.520000
29	43	1.482759
33	48	1.454545
37	53	1.432432
46	64	1.391304
55	75	1.363636
60	81	1.350000
65	87	1.338462
70	93	1.328571
75	99	1.320000

TABLE IV
REC PARAMETERS WITH $a = 0.9$

$A = 0.999$			
r	k_R	n_R	S_R
2	77	95	2.467532
3	78	95	3.653846
4	11	13	4.727273
5	12	13	5.416667
6	29	33	6.827586
7	30	33	7.700000
8	31	33	8.516129
9	32	33	9.281250

$A = 0.9999$			
r	k_R	n_R	S_R
2	78	99	2.538462
3	79	99	3.759494
4	80	99	4.950000
5	11	13	5.909091
6	12	13	6.500000
7	16	17	7.437500
8	20	21	8.400000
9	32	35	9.843750

$A = 0.99999$			
r	k_R	n_R	S_R
2	76	99	2.605263
3	77	99	3.857143
4	4	5	5.000000
5	8	10	6.250000
6	9	10	6.666667
7	24	29	8.458333
8	25	29	9.280000
9	26	29	10.038462

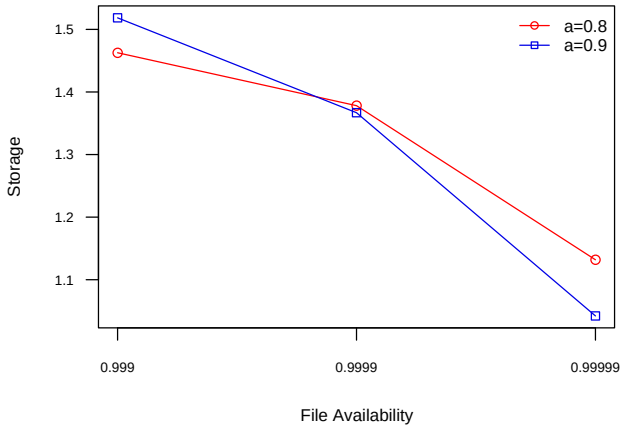


Fig. 2. Storage: REC vs. TEC

the city of Cambridge, MA to monitor temperature, humidity, precipitation, wind speed, pollution level, etc. Each sensor node has an embedded PC running FreeBSD, providing the storage capability of up to 1Gb, wireless communication interface 802.11a/b/g radios, 8.5 dBi omni antennas. Most of the nodes use wireless mesh for the connectivity, while some nodes have connection with the internet, and act as the gateway so that other nodes can be accessed from the Internet.

We chose CitySense as a testbed to implement REC because many nodes of CitySense fail frequently and as such this network presents a practical system setup for testing the correctness and feasibility of our storage scheme. The implemented system provides a web interface and lets people connect to the nodes to distribute their files and retrieve them back when necessary. The erasure code used in our system is Reed-Solomon (5,2), which is capable to tolerate three-server failures and offer three-nine availability.

VI. Conclusions

In archival storage systems where data maintenance is performed much more often than data access, it is desirable to keep the maintenance cost low even if this comes with less efficient data access. In this paper, we investigate a possible extension of erasure coding with this purpose. The presented scheme, REC, requires to contact only two servers to recover from a server's failure, whereas TEC, the traditional erasure coding, would require to contact many more servers and perform an expensive file reconstruction procedure. Therefore REC offers an excellent maintenance efficiency. The tradeoff of REC is that it requires more storage space and a higher cost to access the data. However, as aforementioned, data access is rare for an archival system, and our analysis have shown that the storage space required by REC is albeit larger, but not by a significant margin. This margin is getting smaller if the system is more demanding on either server availability or file availability. In other words, we recommend REC for systems that are stable and that want an excellent file availability.

Acknowledgment

The authors would like to thank EMC Innovation Network, Office of the CTO for sponsoring the project and Prof. Matt Welsh for making CitySense available for the implementation discussed in this paper.

References

[1] R. Murty, G. Mainland, I. Rose, A. Chowdhury, A. Gosain, J. Bers, and M. Welsh, "CitySense: An Urban-Scale Wireless Sensor Network and Testbed", in *IEEE International Conference on Technologies for Homeland Security*, 2008.

[2] B. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. F. Kaashoek, J. Kubiatowicz, R. Morris, "Efficient replica maintenance for distributed storage systems ", in *Proc. of NSDI*, 2006, pp. 45–58.

[3] H. Xia, Andrew A. Chien, "RobuSTore: a distributed storage architecture with robust and high performance ", in *Proc. of SC*, 2007, pp. 1–11.

[4] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer , C. Wells , B. Zhao , "OceanStore: An Architecture for Global-Scale Persistent Storage ", 2000.

[5] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, J. Kubiatowicz, "Pond: The OceanStore Prototype", in *Proc. of FAST*, 2003, pp 1–14.

[6] S. Ghemawat, H. Gobioff, S. Leung, "The Google File System ", in *Proc. of SOSP*, 2003.

[7] R. B. Kiran, K. Tati, Y. Cheng, S. Savage, G. M. Voelker, "Total Recall: System Support for Automated Availability Management ", in *Proc. of NSDI*, 2004, pp 337–350.

[8] F. Dabek, M. F. Kaashoek, D. Karger , R. Morris , I. Stoica, "Wide-area cooperative storage with CFS", in *ACM SIGOPS*, 2001, pp 202–215.

[9] Q. Xin, E. L. Miller, T. J. E. Schwarz, "Evaluation of distributed recovery in large-scale storage systems", in *Proc. of High performance Distributed Computing*, 2004, pp 172–181.

[10] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, R. P. Wattenhofer, "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment", in *Proc. of OSDI*, 2002, pp 1–14.

[11] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. Wainwright and K. Ramchandran, "Network Coding for Distributed Storage Systems", in *Proc. of IEEE INFOCOM*, 2007.

[12] A. G. Dimakis, K. Ramchandran, Y. Wu, C. Suh, "A Survey on Network Codes for Distributed Storage", in *Proc. of IEEE*, 2011.

[13] R. Hasan, Z. Anwar, W. Yurcik, L. Brumbaugh, R. Campbell, "A survey of peer-to-peer storage techniques for distributed file systems", in *Proc. of ITCC*, 2005, 205–213 Vol. 2.

[14] R. Rodrigues and B. Liskov, "High Availability in DHTs: Erasure Coding vs. Replication", in *Proc of IPTPS*, 2005.

[15] R. Bhagwan, S. Savage, and G. Voelker, "Replication Strategies for Highly Available Peer-to-Peer Storage Systems", in *Proc. of MMCN*, 2002.

[16] J. Plank, "Erasure codes for storage applications", in *Tutorial, FAST-2005: 4th USENIX Conference on File and Storage Technologies*, 2005.

[17] H. Weatherspoon and J. Kubiatowicz, "Erasure codes vs. Replication: A quantitative comparison", in *Proc. IPTPS*, 2002.

[18] J. Plank, J. Luo, C. Schuman, " A performance Evaluation and Examination of Open-Source Erasure Coding Libraries for Storage".

[19] S. B. Wicker and V. K. Bhargava, ed., "Reed-Solomon Codes and Their Applications", in *Piscataway, NJ: IEEE Press*, 1983.

[20] B. Gaidioz, B. Koblitz, and N. Santos, "Exploring High Performance Distributed File Storage Using LDPC Codes".

[21] Ashish Khisti, "Tornado Codes and Luby Transform Codes", 2003.

[22] M. Luby, "LT Codes", in *Proc. IEEE Foundations of Computer Science (FOCS)*, 2002.

[23] A. Shokrollahi, "Raptor Codes", in *IEEE Trans. on Information Theory*, vol. 52, pp. 2551-2567, 2006.

[24] J. Bayers, M. Luby, M. Mitzenmacher and A. Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data".

[25] I. Stoica, R. Morris, D. Karger, M. Kaashock, and H. Balakrishman, "Chord: A scalable peer-to-peer lookup protocol for internet applications," in *ACM SIGCOMM*, San Diego, CA, August 2001, pp. 149–160.