Software-based Erasure Codes for Scalable Distributed Storage

Joseph A. Cooley, Jeremy L. Mineweaser, Leslie D. Servi, Eushiuan T. Tsung
MIT Lincoln Laboratory
cooley, jlm, servi, eushiuan @ll.mit.edu

Abstract

This paper presents a new class of erasure codes, Lincoln Erasure codes (LEC), applicable to large-scale distributed storage that includes thousands of disks attached to multiple networks. A high-performance software implementation that demonstrates the capability to meet these anticipated requirements is described. A framework for evaluation of candidate codes was developed to support in-depth analysis. When compared with erasure codes based on the work of Reed-Solomon and Luby, tests indicate LEC has a higher throughput for encoding and decoding and lower probability of failure across a range of test conditions. Strategies are described for integration with storage-related hardware and software.

1. Introduction

In many ways, information is a critical element of operations for the consumer, corporate enterprise, academia, government, and military. Consumers are amassing large collections of digitized multimedia content and accessing it from multiple localities over the Internet. Throughout the corporate enterprise, knowledge management is improving the utilization of intellectual capital by applying structure to aggregated business information. Increasingly, industries derive value from the creation, processing, and management of information. Information plays a central role in the execution of modern warfare as the Information Revolution seen in society reaches military affairs. Current and emerging capabilities in intelligence, surveillance, and reconnaissance will produce large quantities of information that must be rapidly and reliably available. For a diverse class of information consumers, both the volume and the value of information emerge as significant factors that represent major challenges for traditional storage management methods. Contributing factors include demands for wide accessibility, a dynamic marketplace, and the need for a reliable data storage. As needs vary widely, no single product solution satisfies all users. With a scalable architecture for distributed storage systems and a flexible strategy for cost-effective acquisition, operations, administration, and maintenance, valuable information resources can be protected and hence made more reliable without adversely impacting accessibility and performance.

The rest of Section 1 describes how the distributed storage concept emerges from analysis of critical technology trends and the strategies for information protection using erasure code techniques. Section 2 describes related work in erasure coding for distributed storage and a new erasure code, the Lincoln Erasure Code (LEC). The design, implementation, and performance of this code and its integration in the distributed storage system are presented in Section 3. Conclusions are listed in Section 4.

1.1. Critical technology trends

To sustain effectiveness, an architecture must be consistent with the long-term trends in technology development. Due to the exponential growth of performance per dollar spent in silicon transistor density, optical fiber bandwidth, and stored bits per square inch, execution of this strategy has proven difficult. However, the technologies are now positioned to enable a scalable distributed storage architecture that can be tailored to meet user needs.

Silicon transistor density doubling, known as Moore's Law, has persisted for decades. The doubling period of 18 months is expected to continue at this rate for another 15 years. A similar trend has emerged that tracks the increase in storage density for the magnetic media in hard disk drives. In this case, the doubling period is 12 months, surpassing that of Moore's Law. In the last decade, the commercial growth of optical fiber

This work is sponsored by the United States Air Force under Air Force Contract #F19628-00-C-0002. Opinions, interpretations, recommendations and conclusions are those of the authors and are not necessarily endorsed by the United States Government.

networking technology has repeatedly increased the information carrying capacity of a single fiber at an exponential rate which exceeds those for both transistor and storage density. Recently the fiber capacity doubling period has been measured as nine months. In the future, system architectures established before the emergence of low-cost, high-speed optical networks may struggle to keep pace with new architectures. As the leading critical technology trend, optical networking represents the foundation of a scalable distributed storage architecture [1].

1.2. Distributed storage systems

The relative performance of storage, memory, computing, and communication has a profound impact on storage architecture design. In traditional network storage models, individual users equipped with commodity network interfaces cannot fully utilize the system backplane (communications link) with storage transactions. In many current storage systems, only a small number of disk drives are involved in storing the contents of an individual computer file. Most personal computers include a single high capacity hard disk drive to which read and write requests are sent. For higher performance and reliability, business systems often employ Redundant Array of Inexpensive Disks (RAID) controllers to stripe data across multiple disks and generate parity check information. [2]. For improved reliability, these controllers as well as software RAID implementations can mirror users' data to a second identical array and/or generate parity-check information. The distributed storage system concept emerges from analysis of the critical technology trends seen in Figure 1. Extrapolating the trends in optical networking and data storage to 2010 suggests that thousands of hard disks running in parallel will be required to keep pace with COTS communication links operating at many terabits per second.

1.3. Strategies for information protection

In addition to addressing these performance challenges, a scalable storage architecture must provide an effective means to protect the valuable information that is entrusted to storage systems. In the simple mirroring and parity schemes described above, administrators of an operational system must treat any single failure as an emergency or risk exposing users to service disruptions and information loss. While geographically distributed data centers can improve overall system availability, the corresponding need for emergency response personnel located at each data center increases the cost of system operation.

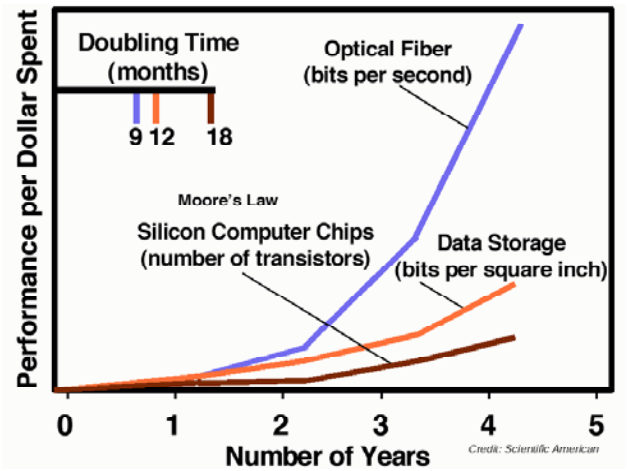

Figure 1. Critical technology trends.

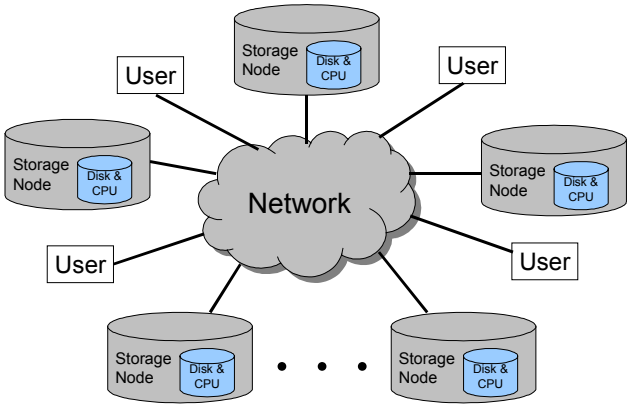

Figure 2. Distributed storage system architecture.

In a distributed storage system, an end user's computer interacts directly and concurrently with a potentially large number of networked storage nodes as shown in Figure 2. These nodes include CPU, main memory cache, and persistent storage on hard disk drives. The nodes receive and respond to user requests across the network. On the end user's computer, information is cached in main memory. User files are stored to the system as a set of fragments, which may include parity information for erasure correction. The number of fragments can be determined by requirements for performance and/or reliability.

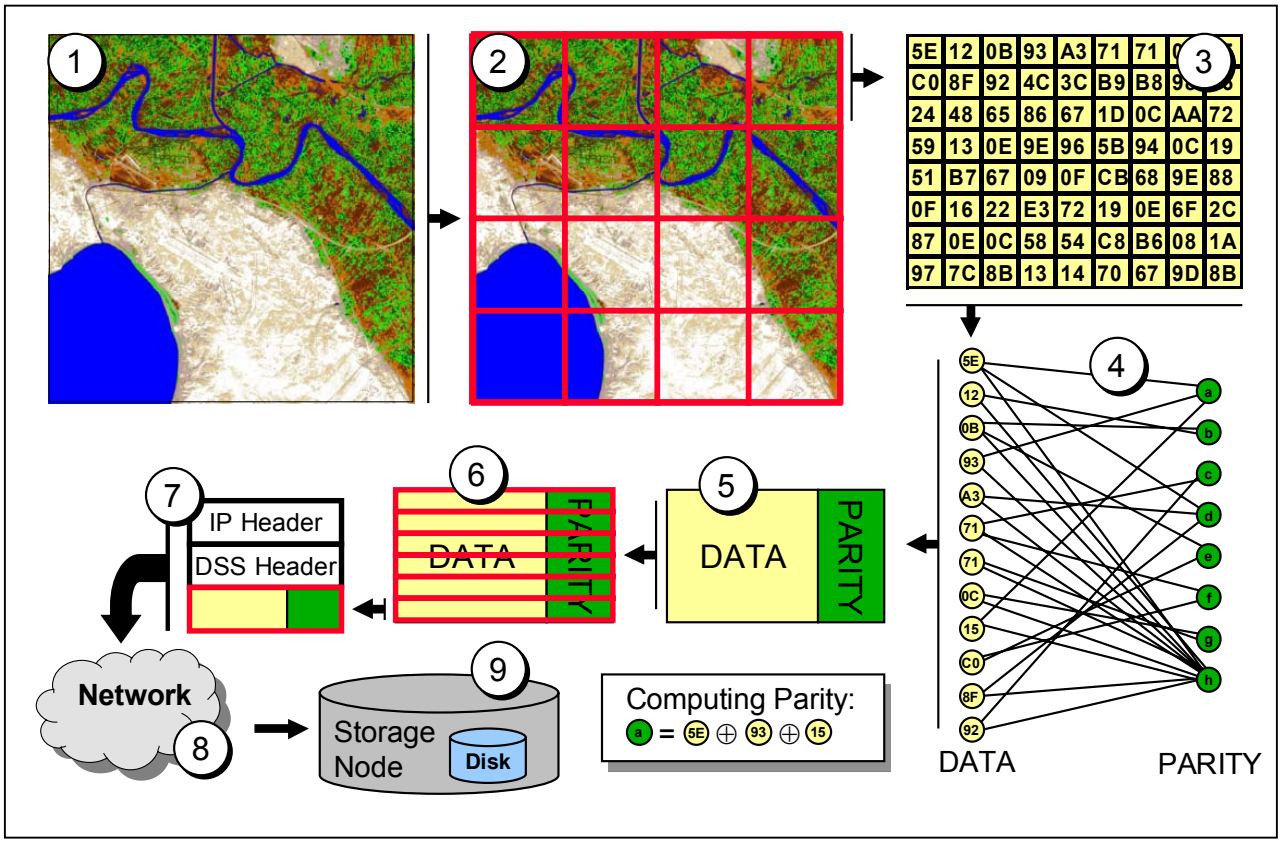

Figure 3. Encoding data for distributed storage.

2. Erasure coding for distributed storage

Erasure recovery is necessary when the failure of a communications channel or storage device prevents the direct retrieval of a previously stored data file. Although such failures are often caused by inherent properties of the device or channel, they may also be the result of malicious or careless actions. In general, solutions to this longstanding problem involve the construction of a larger file with explicit or implicit redundancy of information. After an erasure (i.e. the loss of a bit), it is this redundancy which enables successful data recovery. One common, simple approach is to create one or more copies of the original data. This results in a level of added protection far below that of alternative methods employing the same amount of redundancy.

Figure 3 shows an example of encoding data for distributed storage. A file is broken up into a number of fragments (step 2) and encoded (step 3) as described in the next section. A graph of parity and data nodes is created. Each parity node is computed by performing the exclusive or of each data node to which it is connected (steps 4 and 5). The data and parity fragments are then distributed to different storage nodes over the network (steps 6 through 9).

2.1. Related work

Erasure codes have been applied to distributed storage in the OceanStore project at the University of California Berkeley [3]. Both Reed-Solomon and Tornado codes (described below) were implemented in software to improve the availability of archived data. With both of these codes, throughput is limited by the high computational costs of encoding and decoding. Reed-Solomon codes are further limited by scaling difficulties for use with more than 255 storage elements. As technology trends begin to require thousands of fragments to saturate high-speed network links, alternative erasure codes may prove more effective.

The Koh-i-Noor project at Microsoft seeks to construct high-capacity storage systems with high reliability at low cost [4]. The motivation is to reduce total cost of ownership for enterprise-class storage systems such as Hotmail. Through the use of erasure codes, system maintenance requirements can be significantly reduced

while preserving the integrity of stored information for long periods of time.

The Reed-Solomon code, a special case of the Bose-Chaudhuri code, supports both erasure correction and error correction. It offers optimal efficiency such that any available parity element can be substituted for any erased data element in the block. Parity-check information is generated through operations on a Galois field [5]. The computational cost of this process is related to the size of the field, where typical Reed-Solomon implementations operate in a field of size 2^8, or one with 255 elements. For this code, a linear increase in the depth of the field results in an exponential increase in computational cost, making it impractical for use in storage systems requiring thousands of fragments. Two advantages of Reed-Solomon are storage efficiency and deterministic behavior.

One approach to large block coding is to segment the data into a number of "mini-blocks" and separately apply Reed-Solomon coding to each mini-block. While high throughput can be achieved for encoding and decoding, this is less space-efficient than other alternatives.

In 1960, R. Gallager's Sc.D. thesis [6] proposed a new approach to the erasure code problem using a *bipartite graph*, illustrated in Figure 3, step 4. This type of graph contains two sets of nodes with edges between nodes of different sets, but no edges between nodes within the same set. The key idea is to create a graph with a set of nodes corresponding to the data bits and a set of nodes corresponding to the parity bits and connect some of the data nodes to the parity nodes. The encoding process consists of setting a parity node equal to the exclusive or (XOR) of the data nodes to which it is connected. Gallager's approach is based on a *regular graph* in which all nodes of each type have the same number of edges. In this code, the number of edges is proportional to the number of nodes. Since the resulting edge count is small compared to the number of edges in a fully connected bipartite graph, these are called low density parity check (LDPC) codes.

In 1997, Luby et. al [7] proved that LDPC codes have improved properties if the number of edges connected to each node varies from node to node. For such *irregular* codes, the challenge is to characterize the degree of irregularity that yields the best coding performance. This is accomplished in two stages. First, a probability distribution is specified for the degree of a parity node. A distribution may also be specified for the degree of a data node. Then a procedure is developed for realizing an effective code based on the distribution. In contrast to Reed-Solomon codes, the Luby Transform code [8] and related codes [7] (including Tornado codes [9]) provide *probabilistic* erasure correction through an iterative decoding algorithm. This class of codes provides a significant reduction in computational cost for encoding and decoding, but it sacrifices some storage efficiency in the process.

The Luby Transform code specifies a probability distribution for the number of data nodes connected to each parity node. As edges between nodes are defined, the code specifies that all eligible combinations of data nodes may be chosen with equal probability.

A file can be multicast to thousands of users using the Luby Transform code by sending a small amount of data and a stream of parity information. The value of each transmitted parity node is computed as the XOR of the data nodes to which it is connected. The Luby code is well suited to this application because it is not necessary to select (and generate) a specific (and limited) quantity of parity check information prior to transmission. Rather, the transmission simply continues until the minimum required quantity of information has been exchanged. This property is advantageous for multicast applications; in persistent data storage applications, however, storing only this minimal quantity of data and parity information would adversely impact system reliability.

2.2. Lincoln erasure code (LEC)

In contrast to the Luby code which consists of predominately parity information, LEC saves *all* of the data. However, like the Luby code, LEC's parity information is generated using an irregular bipartite graph. As such, the number of parity nodes of degree *j*, $N_j(s, p, k, A, j^*, imax)$, in LEC must be specified. In LEC, for $j = j^*+2, j^*+3, \ldots, imax$,

$$N_j(s, p, k, A, j^*) = s\left(\frac{Lk}{j - j^* - 1} - \frac{Lk + 1 - A\sqrt{Lk}}{j - j^*}\right)$$

where *s* is a scaling factor, *p* is the fraction of corruption being protected against, *k* is the number of data nodes, *imax* is the maximum possible degree, *j** is two less than the minimum degree, and *A* is a variable.

For a given target reliability, desired loss protection, *L*, and number of data characters, *k*, a preferred set of parameters must be empirically found for *s*, *A*, *j**, and *imax*. In addition, the above formula must be perturbed to account to its possible non-integer values. For each set of parameters a specific graph must be generated by randomly selecting which data nodes are connected to which parity nodes. To encode using LEC, all data is used along with parity information computed from the XOR of the data nodes connected to it (as is performed for the Gallager and Luby code). To decode, erasures are corrected with a decoder that is modeled after that of Luby's code.

3. Evaluation process and performance results

A software evaluation tool was developed to study the reliability and throughput of large block LDPC codes. The software performs encoding, decoding, generation of random codes (based on parameters that determine the degree distribution of parity-check elements), and reporting of performance metrics for reliability and throughput. The core function of the tool is the detailed examination of the decoding process (erasure correction) under a variety of loss conditions. In this mode, parity and data exist abstractly but do not have actual values. Instead, the tool simply tracks the erasure status of each element during the decoding process.

For LEC, one can select a desirable set of parameters given the block size and the block failure probability that is required when a given fraction of the block is erased. Since the number of parameters is small, desirable codes can be found by trial and error or using a search algorithm. A software implementation of the search algorithm partially automates the parameter selection process. Given the block size (number of data elements and parity elements) and the fraction of the block erased, a parameter search is performed to identify the parameters that achieve the highest reliability.

In the next step, specific codes are generated and tested using the selected parameters. The graph that exhibits the highest reliability is saved until a better code is found. Since the reliability of random codes generated with the same parameters varies by as much as a factor of three, this process is repeated many times (i.e. 100). Encoding and decoding throughput is measured for graph codes that demonstrate high reliability.

3.1. Design and implementation

Generated graphs are formatted in a parity-centric tabular view. Each parity element identifier is followed by a list of the data elements to which it is connected. For throughput analysis, a data-centric view of the graph is also stored. In this view, each data element is followed by a list of the parity elements to which it is connected. In both, the node listings are sorted to improve locality during encoding and decoding.

In addition to reliability, encoding/decoding throughput for actual files was also a matter of interest. Several coding methods were compared for a variety of file sizes. Each test file was segmented into a series of fixed-size blocks for use with the codec (coding/decoding). During the decoding process, throughput can be increased with an optional journaling technique. During block decoding, correction actions are recorded in a journal and indexed according to erasure pattern. If a subsequent block contains the same erasure pattern (or a subset) as a previous block, then journal results can be used. Otherwise, an iterative decoding process is performed.

3.2. Performance results

In this section, LEC performance results will be presented for several target reliabilities, percent losses, and block sizes. First, the case of targeting 10^{-6} reliability for 20 percent loss and 5040 data elements will be presented in detail.

Using the search algorithm, it was determined that 1760 parity elements was needed to achieve 10^{-6} reliability at 20 percent loss for 5040 data elements. The set of parameters selected was A=1, imax=2229, j*=10, s=1.43. These parameter values were inserted in the distribution formula to generate the parity node degree distribution. A graph code consisting of 5040 data 8-byte elements and 1760 parity 8-byte elements was then constructed. After randomly "erasing" 20 percent of the data and parity elements, an attempt was made to decode the unknown data elements. Table 1 compares the throughput of various erasure codes for encoding and decoding a 13.1 MByte file. Tests were conducted on a PC with a 1.7 GHz Intel Xeon CPU. The table shows that LEC encodes and decodes faster than both the Luby and Reed-Solomon methods.

Table 1. Throughput of erasure codes.

Erasure Coding Method	Encoding (Mbps)	Decoding (20% Loss) (Mbps)
LEC	240	760
Luby Code	180	220
Reed-Solomon	0.09	0.20

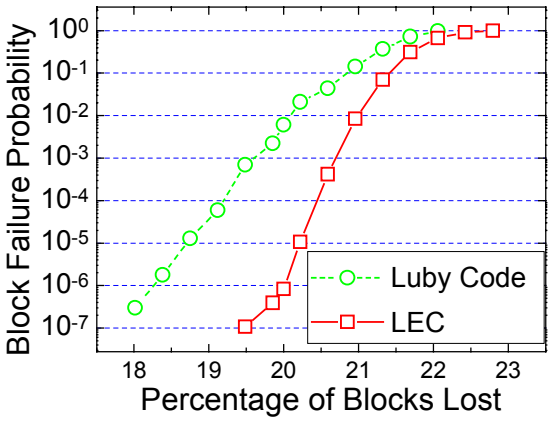

Figure 4. Reliability of erasure codes.

Table 2. 5040 data, targeting 10^{-6} reliability.

Percent Loss	# Parity Elements, p	LEC Reliability	Luby Reliability	Minimum # Parity Elements Needed, p'	Overhead, p / p'
1	130	6.5×10^{-7}	1.0	51	2.55
10	860	3.3×10^{-7}	6.1×10^{-1}	560	1.54
20	1760	8.3×10^{-7}	6.1×10^{-3}	1260	1.40
50	6350	1.7×10^{-6}	3.5×10^{-6}	5040	1.26
75	20000	2.2×10^{-6}	2.8×10^{-7}	15120	1.42

Table 3. 5040 data, targeting 10^{-4} reliability.

Percent Loss	# Parity Elements, p	LEC Reliability	Luby Reliability	Minimum # Parity Elements Needed, p'	Overhead, p / p'
1	110	1.4×10^{-4}	1.0	51	2.16
10	810	2.3×10^{-5}	1.0	560	1.45
20	1700	3.5×10^{-5}	2.9×10^{-2}	1260	1.35
50	6230	9.9×10^{-5}	1.7×10^{-5}	5040	1.24
75	19250	1.1×10^{-4}	7.5×10^{-7}	15120	1.27

Table 4. 2520 data, targeting 10^{-6} reliability.

Percent Loss	# Parity Elements, p	LEC Reliability	Luby Reliability	Minimum # Parity Elements Needed, p'	Overhead, p / p'
1	80	1.9×10^{-6}	1.0	25	3.20
10	470	2.3×10^{-6}	9.0×10^{-1}	280	1.68
20	950	1.0×10^{-6}	3.2×10^{-2}	630	1.51
50	3360	7.3×10^{-7}	6.8×10^{-6}	2520	1.33
75	10400	1.8×10^{-6}	1.1×10^{-6}	7560	1.37

Table 5. 2520 data, targeting 10^{-4} reliability.

Percent Loss	# Parity Elements, p	LEC Reliability	Luby Reliability	Minimum # Parity Elements Needed, p'	Overhead, p / p'
1	65	1.4×10^{-4}	1.0	25	2.60
10	435	1.4×10^{-4}	1.0	280	1.55
20	900	5.8×10^{-5}	2.9×10^{-1}	630	1.43
50	3240	3.6×10^{-5}	2.6×10^{-4}	2520	1.29
75	10100	2.0×10^{-4}	1.8×10^{-6}	7560	1.34

Figure 4 compares the reliability of LEC generated with the above parameters and a Luby code of the same length. At the 20 percent loss point (for which LEC was optimized), the block failure probability of LEC is 8×10^{-7} as compared to 6×10^{-3} for the Luby code. Each point on the plot was experimentally generated using the software evaluation tool. The uncertainty in the probabilities is half an order of magnitude. Since the number of encoding/decoding runs necessary to produce statistically meaningful reliability results increases as the reliability becomes better, it is not computationally feasible to experimentally generate block failure probabilities lower than 10^{-7}. Probabilities lower than 10^{-7} can be extrapolated from the plot. Note that LEC outperforms the Luby code across the entire range of percent losses tested.

An equivalent Reed-Solomon code would detect all erasures up to the number of parity elements for the range of losses in Figure 4. However, as seen in Table 1, the encoding and decoding speed is so slow that it is not a realistically feasible code to use.

Tables 2 through 5 present results for varying target reliabilities, percent losses, and block sizes. For practical purposes, a distributed storage system would most likely be concerned with lower percent losses (i.e. less than 20 percent), however we present a wider range of results for completeness.

Table 2 shows the number of parity elements needed for different percent losses when targeting 10^{-6} reliability for 5040 data elements. For each percent loss, the software evaluation tool was used to determine the number of parity elements, p, and the set of parameters that yields a desirable graph code. This graph code was then used to generate the LEC reliability at that percent loss. Note that the LEC reliability in Table 2 should match the target reliability of 10^{-6}. For example, to achieve 10^{-6} reliability when losing 10 percent of the blocks with 5040 data elements requires a graph code consisting of 860 parity elements. Using fewer than 860 parity elements will degrade reliability.

LEC and Luby codes were also compared for the different percent losses. Table 2 shows that LEC outperforms the Luby code for losses of 1, 10, and 20 percent. Performance of the two codes is similar for 50 percent loss. However, for low percent losses, it must be noted that LEC and Luby codes were designed for slightly different applications as stated in Section 2.1.

The fifth column of Table 2 specifies the minimum number of parity elements required for successful decoding. The process will be unsuccessful if fewer than this number of parity elements are available. This limit is computed by solving the following equation

$$(p' - d)\ (1 - L) >= d$$

for p', the minimum number of parity elements needed, where d is the number of data elements and L is the percent loss. The last column in Table 2 shows the overhead required to achieve the target reliability. Overhead is defined as the ratio of the number of parity elements used for correction, p, to the number of data elements that are erased (which is equivalent to p'). This overhead steadily decreases as the percent loss increases from 1 to 50 percent and slightly increases for 75 percent loss.

Table 3 shows the number of parity elements needed for different percent losses when targeting 10^{-4} reliability for 5040 data elements.

Tables 4 and 5 show the number of parity elements required for different percent losses with 2520 data elements when targeting 10^{-6} and 10^{-4} reliability. With a 50 percent decrease in the number of data elements, the number of parity elements will decrease by approximately half.

Tables 2 through 5 demonstrate the tunability of LEC. By specifying a data percent loss that the distributed storage system would like to protect against and the reliability it hopes to achieve, the search algorithm can find the number of parity elements necessary to achieve the requirements. Likewise, if cost is a factor in designing the storage system, the user can specify the number of parity elements it wants to use at a certain percent loss. The search algorithm can then determine the level of reliability it can achieve with this number of parity elements.

The next section briefly introduces a system architecture that incorporates this coding technique to meet users' reliability requirements while providing high performance storage services.

3.3. System architecture and integration

Erasure coding offers important capabilities for storage system architectures. As device-level storage density and read/write parallelism increase, excellent scalability can be maintained with a hardware implementation of the Lincoln Erasure LDPC code. Already, the high throughput of the software-based codec enables system development with inexpensive COTS components. This provides fault tolerance, higher reliability, and space-efficiency in comparison to pure replication and mirrored hardware. A software-based code can also protect local resources such as a RAID or a storage area network (SAN).

The storage node is the fundamental building block of the architecture. Each node contains multiple hard disk drives; the number of drives is determined by overall system capacity requirements. RAID 0 disk arrays may be created at each node to reduce disk seek latency and increase read/write throughput. Processing resources at each node can be applied to coding tasks or other data processing as directed by users. For certain classes of parallel applications, including image processing and data mining, significant speedup can be achieved by performing data-intensive computations "near" the storage location and sending only results to users [10].

The network interface card in each storage node provides connectivity to the user community and the rest of the storage system. High read/write throughput is achieved by aggregating many storage nodes on an optical network. For example, although data rates may reach 40-100 Gbps within the core of the network, individual storage nodes can be attached with low-cost Gigabit Ethernet adapters. With a sufficient number of storage nodes, the full capacity of the core network can be exploited entirely by a single user or shared among multiple users.

The most robust network topology is one in which each node is directly connected to every other node. Such networks are said to have an exponential topology because the total number of connections increases exponentially with a linear increase in the number of nodes. Any single link failure causes minimal disruption since traffic can be rerouted through an intermediate node. For large

networks such as the Internet, an exponential topology is not feasible. The scale-free topology is a cost effective alternative in which small networks are aggregated into larger networks. This structure is repeated at multiple levels to form a network hierarchy. In this case, a single link failure at a high-level aggregation point can separate the nodes into disjoint sub-networks and prevent communication among users [11].

In conjunction with a distributed storage architecture, erasure coding techniques insulate users from network outages and node failures. Although such problems are promptly detected and reported to administrators, events do not escalate into emergencies requiring immediate response. Instead, user needs will be applied to determine an appropriate level of protection for each item stored in the system. A simple, high-performance software implementation of the erasure coding technique enables any user to select appropriate parameters and interact with the storage system.

In many cases, this storage architecture can reduce or eliminate the need for an offline backup system. Upon detection of node or network failures, a stored file can be retrieved and decoded using the erasure code. Then the file can be encoded and redistributed to the accessible nodes. In contrast to systems based on serial backup media such as magnetic tapes, this system provides direct random access to all stored files with minimal contention among multiple users. Overall, the mitigation of offline backup requirements results in cost savings for infrastructure and personnel and improves user satisfaction with the storage service.

In addition, system integration of the erasure code yields economic benefits from the device to the inter-network level. In particular, graceful degradation of the storage service under failures and attacks greatly relaxes system maintenance schedules. Failed hardware can be restored or replaced along with planned upgrades according to a periodic schedule. At the same time, the erasure code significantly reduces storage capacity requirements relative to pure replication [12]. These factors translate directly into monetary savings for system operators and users.

4. Conclusions

Extrapolating critical technology trends in networking, storage, and processing into the future indicates that the most scalable storage systems will be established on a firm foundation of networking. The disparity between optical fiber carrying capacity and hard disk drive throughput suggests that thousands of disk drives may be needed to fully utilize future network links. Existing RAID approaches solely based on mirroring and parity-check schemes do not provide cost-effective solutions to these challenges.

Compared to Reed-Solomon, Tornado, and Luby codes, LEC provides higher performance both in terms of throughput and reliability and greater scalability in a flexible software implementation except for very high percentage of loss. It is also tunable to system requirements for the level of reliability or cost of the storage system. It enables cost-effective implementations of large, highly distributed, reliable storage systems.

References

[1] Stix, G. "The Triumph of the Light." *Scientific American*, January 2001, 80-86.

[2] Patterson, D., Gibson, G., and Katz, R. "A Case for Redundant Arrays of Inexpensive Disks (RAID)." In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, June 1988, 109-116.

[3] Kubiatowicz, J., et. al. "OceanStore: An Architecture for Global-Scale Persistent Storage." In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000, 190-201.

[4] Isard, M., Manasse, M., Thekkath, C. Koh-i-Noor. [November 2002] http://research.microsoft.com/research/sv/kohinoor/

[5] Peterson, W. W. *Error Correcting Codes*. The MIT Press, Wiley and Sons, 1961.

[6] Gallager, R. G. "Low Density Parity Check Codes." Sc.D. Thesis. Department of Electrical Engineering. MIT. Cambridge, Massachusetts, 1960.

[7] Luby, M., Mitzenmacher, M., Shokrollahi, M.A., Spielman, D., Stemann, V. "Practical loss-resilient codes." In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997, 150-159.

[8] Luby, M. *Lost Packet Recovery Method for Packet Transmission Protocols*. WIPO Patent WO 00/18017, 30 March 2000.

[9] Byers, J., Luby, M., Mitzenmacher, M. "Accessing Multiple Sites in Parallel: Using Tornado Codes to Speed Up Downloads." In *Proceedings of IEEE INFOCOM 1999*, 275-283.

[10] Reidel, E., Faloutsos, C., Gibson, G.A., Nagle, D. "Active Disks for Large-Scale Data Processing." *IEEE Computer*, June 2001, 68-74.

[11] Albert, R., Jeong, H., Barabási, A.-L. "Error and attack tolerance of complex networks." *Nature* 406, 2000, 378 - 382.

[12] Weatherspoon, H., Kubiatowicz, J. "Erasure Coding vs. Replication: A Quantitative Comparison." In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS 2002)*, March 2002, 328-338.