In Search of I/O-Optimal Recovery from Disk Failures

Osama Khan, Randal Burns
Department of Computer Science
Johns Hopkins University
{*okhan, randal*}*@cs.jhu.edu*

James Plank
Department of Electrical Eng. and Comp. Science
University of Tennessee
plank@cs.utk.edu

Cheng Huang
Microsoft Research
Cheng.Huang@microsoft.com

Abstract

We address the problem of minimizing the I/O needed to recover from disk failures in erasure-coded storage systems. The principal result is an algorithm that finds the optimal I/O recovery from an arbitrary number of disk failures *for any* XOR-based erasure code. We also describe a family of codes with high-fault tolerance and low recovery I/O, e.g. one instance tolerates up to 11 failures and recovers a lost block in 4 I/Os. While we have determined I/O optimal recovery for any given code, it remains an open problem to identify codes with the best recovery properties. We describe our ongoing efforts toward characterizing space overhead versus recovery I/O tradeoffs and generating codes that realize these bounds.

1 Introduction

Recovery from failures has become a critical component of disk storage systems because they have reached such a massive scale that failures must be expected and dealt with as a matter of regular operation [8]. Large scale deployments now typically tolerate multiple failures both to keep service available and to avoid data loss, e.g. three replicas has become the de facto standard in Hadoop! and systems utilizing RAID-6 are widely deployed.

We answer a fundamental question in recovery performance: what are the fewest number of I/Os needed to recover from an arbitrary number of disk failures? As the amount of redundancy grows, storage system codes offer many different schedules to recover a lost disk that vary widely in their I/O requirements. For example, in row-diagonal parity [23] and Even-Odd codes [22] that tolerate two disk failures, I/O can be reduced by 25% by recovering a combination of rows and diagonals that share blocks, rather than the standard practice of recovering each row independently. We provide an algorithm that minimizes the I/O recovery cost for any XOR-based code.

Two application contexts, cloud storage systems and deep archival storage, motivate the need for I/O efficient coding (Section 2). Cloud storage systems perform erasure decoding when recovering from disk failures and *when performing system upgrades*. Upgrades occur frequently—they are often continuous [7]—and minimizing I/O limits performance degradation. Deep archival stores include data that are almost never read, but need to be stored for regulatory or archival purposes. For these data, the only workload is introspection and recovery. Therefore, highly fault-tolerant recovery I/O efficient codes allow us to increase scale and save power.

Many advances have been made in improving recovery performance in disk redundancy coding. These include hardware to minimize data copying [6], load-balancing recovery among disks [12], recovering popular data first to decrease read degradation [21], and only recovering blocks that contain live data [19]. Recently, the issue of minimizing I/O recovery schedules has emerged as a research topic. The results for Even-Odd codes [22] and row-diagonal parity [23] represent solutions for two specific codes. We present an algorithm that defines the I/O lower bound for any matrix code and allows multiple codes to be compared for I/O recovery cost.

Optimizing the recovery I/O of existing erasure codes shows benefits, but does not transform recovery. In contrast, codes designed specifically for recovery radically reduce I/O. Our algorithm applied to Liberation codes and Cauchy Reed-Solomon codes reduce I/O by 20-30%: on the same order as previous results [22, 23]. We present a family of codes (Section 4) that leverage the constrained data dependencies of Weaver codes [10] and 2-dimensional properties of Grid codes [15]. One instance of these codes tolerates up to 11 failures and can recover a lost encoded block in 4 I/Os! A Reed-Solomon code with similar properties uses 12 I/Os.

The design of recovery I/O optimal codes remains an open problem. We conjecture that there are tradeoffs between recovery I/O and storage efficiency, i.e. that an increase in storage can reduce I/O at a given fault tolerance. We are pursuing the fundamental bounds for this problem. At the same time, we are exploring the structure of recovery I/O by searching for the best feasible codes using our optimization algorithm.

Note: Regenerating codes provide minimal recovery bandwidth and storage overhead [5]. They were designed for distributed systems in which wide-area bandwidth limits recovery performance. They achieve minimum bandwidth by transferring a smaller amount of data from as many shares of the data as are available. For storage systems, minimizing I/O is more valuable than minimizing bandwidth and regenerating codes that access all existing shares of data increase I/O.

2 Applications of I/O-Optimal Recovery

Cloud File Systems: Cloud storage systems, such as Amazon S3 and Windows Azure Storage, assemble massive amounts of unreliable hardware and rely on software to deliver highly reliable and available storage services. Typically, they store three replicas [8] to guard against failures. Erasure coding provides an alternative that improves fault tolerance at reduced storage costs [1].

Cloud storage uses erasure decoding when recovering from failures and, more frequently, when storage nodes are unavailable. Scheduled events, such as patches and software updates, and unscheduled events, such are reboots, make nodes unvailable. For example, an update of the storage software stack rolls out in phases. Small batches of the storage nodes are suspended and the update applied. Then, the entire system is left running until performance metrics stabilize. The entire update process can last hours [7]. During updates, read requests to the unavailable nodes invoke erasure decoding and recovery dictates overall I/O performance.

Deep Archival Stores: Regulatory requirements and preservation dictate that data needs to be archived for future availability. However, a large fraction of this data will never be read. The workload for these systems consists of introspection, checking that data are intact, and error recovery. Pergamum [20] defined archival systems of this type based on massive arrays of idle disks (MAID). They demonstrate that 95% of disks may be powered off at all times. We extend Pergamum's vision of infrequent error detection and look to employing untrusted cloud storage. To increase power savings, we take a much more passive approach to introspection and recovery. Encoding data with large amounts of redundancy allows for the lazy detection of failed devices/sites and recovery from multiple drive and latent sector errors. Combining I/O-efficient erasure coding with secure auditing for outsourced data [4] enables cost reduction in cloud archives.

3 Finding I/O Optimal Recovery Schedules

Any erasure code based on exclusive-or operations may be represented by a bit-matrix-vector product as in Figure 1. A vector of k data bits is multiplied by a $(n \times k)$ Generator matrix to yield an n-element vector called the *codeword*. In our simplified example, each bit (or row) of the input data vector, and consequently the codeword, can represent one or more disk sectors. The code represented in Figure 1 is a RAID-6 code for a four-disk system, where each disk stores two bits (or rows), of the codeword. All XOR-based codes can be represented by a Generator matrix. The difference between the various codes lies in different Generator matrices, and different ways to store the bits on different disks. For example, Greenan *et al* define a "flat" code as one where each bit is stored on a different disk [9]. Thus, Figure 1 could represent a flat code for an 8-disk system.

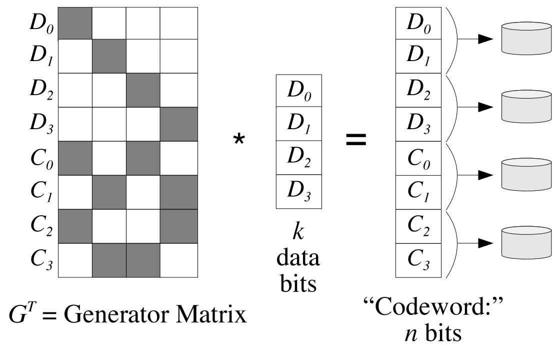

Figure 1: Erasure-coding as a matrix-vector product.

Each bit in the codeword is represented by a row of the Generator matrix. When data is lost, the standard methodology for reconstruction is to create an invertible $(k \times k)$ matrix from k rows of the Generator matrix that correspond to surviving bits in the codeword. This matrix is inverted, and multiplying the inverse by the surviving bits yields the original data [16, 11].

While this technique is general-purpose, it produces one of the many possible ways to reconstruct the lost data. We solve the problem of determining how to recalculate the lost data while minimizing the total number of surviving bits that are read. With each bit representing one or more sectors on disk, minimizing the bits read will minimize the number of disk I/O's required for recovery.

We present an algorithm for this task that is computationally expensive, but feasible for systems of sizes typically used today. In practice, one calculates the recovery strategies for all potential failure scenarios *a priori* and stores them for later use.

We use the code in Figure 1 as an illustrative example. Consider a collection of bits in the codeword whose corresponding rows in the Generator matrix sum to zero. One example is D_0, D_2 and C_0. We call such a collection of bits a *decoding equation*, because the fact that their sum is zero allows us to decode any one of its bits as long as the remaining bits are not lost. For example, if D_2 is lost, and both D_0 and C_0 are not, then this equation may be used to decode D_2.

Suppose that we enumerate all decoding equations for a given Generator matrix, and suppose some subset F of the codeword bits are lost. Then for each bit $f_i \in F$, we determine the set E_i of decoding equations for f_i. Formally, an equation $e_i \in E_i$ if $e_i \cap F = \{f_i\}$. Our goal is to select one equation e_i from each E_i such that the number of elements in the union of all e_i is minimized.

For example, suppose bits D_0 and D_1 in Figure 1 are lost. A standard way to decode the failed bits is to use coding bits C_0 and C_1. In equation form, $F = \{D_0, D_1\}$, $e_{D_0} = \{D_0, D_2, C_0\}$, and $e_{D_1} = \{D_1, D_3, C_1\}$. Since e_{D_0} and e_{D_1} have distinct elements, their union is composed of six elements, which means that four are required for recovery. However, if we use $\{D_1, D_2, C_3\}$ for e_{D_1}, then $|e_{D_0} \cup e_{D_1}|$ is five elements, meaning that three are required for recovery. This saves one I/O operation.

Thus, our problem is as follows: Given $|F|$ sets of decoding equations $E_0, E_1, \ldots E_{|F|-1}$, we wish to select one equation from each set such that the size of the union of these equations is minimized. Unfortunately, this problem is NP-Hard in $|F|$ and $|E_i|$.[1] However, we can solve the problem for practical values of $|F|$ and $|E_i|$ by converting the equations into a directed, weighted graph and finding the shortest path through the graph.

Given an instance of the problem, we convert it to a graph as follows. First, we represent each decoding equation in set form as an n-element bit string. For example, $\{D_0, D_2, C_0\}$ is represented by `10101000`.

Each graph node is also represented by an n-element bit string. There is a starting node Z whose string is all zeroes. The remaining nodes are partitioned into $|F|$ sets labeled $S_0, S_1, \ldots S_{|F|-1}$. Each node in S_i is at the same depth (number of edges) relative to Z as any other node in S_i. For each equation $e_0 \in E_0$, there is a node $s_0 \in S_0$ whose bit string equals e_0's bit string. There is an edge from Z to each s_0 whose weight is equal to the number of ones in s_0's bit string.

Traversing a single level (or edge) in the graph signifies the recovery of a single bit in F. For each node $s_i \in S_i$, there is an edge that corresponds to each $e_{i+1} \in E_{i+1}$. This edge is to a node $s_{i+1} \in S_{i+1}$ whose bit string is equal to the bitwise OR of s_i and e_{i+1}'s bit strings. The OR calculates the union of the equations leading up to s_i and e_{i+1}, with s_{i+1} denoting the cumulative number of elements required for recovery up to that point. The weight of the edge is equal to the difference between the number of ones in s_i and s_{i+1}'s bit strings. The shortest path from Z to any node in $S_{|F|-1}$ denotes the minimum number of elements required for recovery. If we annotate each edge with the decoding equation that creates it, then the shortest path contains the equations that are used for recovery.

To illustrate, suppose again that $F = \{D_0, D_1\}$, meaning $f_0 = D_0$ and $f_1 = D_1$. The decoding equations for E_0 and E_1 are enumerated below:

E_0	E_1
$e_{0,0}$ = `10101000`	$e_{1,0}$ = `01010100`
$e_{0,1}$ = `10010010`	$e_{1,1}$ = `01101110`
$e_{0,2}$ = `10011101`	$e_{1,2}$ = `01100001`
$e_{0,3}$ = `10100111`	$e_{1,3}$ = `01011011`

These equations may be converted to the graph depicted in Figure 2, which has two shortest paths of length five: $\{e_{0,0}, e_{1,2}\}$ and $\{e_{0,1}, e_{1,0}\}$. Both require three bits for recovery: $\{D_2, C_0, C_3\}$ and $\{D_3, C_1, C_2\}$.

While the graph clearly contains an exponential number of nodes, one may program Dijkstra's algorithm to determine the shortest path and only create the graph on demand. For example, in Figure 2, the dotted edges and grayed nodes will not be constructed, because the shortest path is discovered before nodes `10011101` and `10100111` are evaluated by the algorithm.

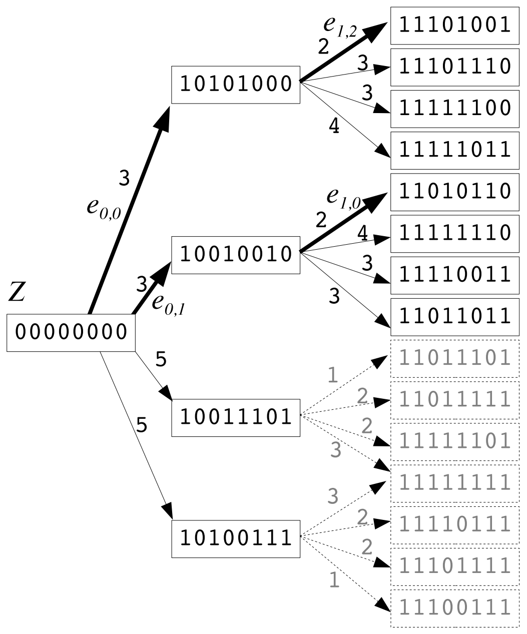

Figure 2: Recovery graph when D_0 and D_1 are lost.

[1] Adam Buchsbaum, personal communication, reduction from Vertex Cover.

Greenan *et al.* [9] use a similar approach to enumerate the recovery equations for flat-XOR codes. Their algorithm employs pruning heuristics on the search space, rather than converting the problem into a graph.

Figure 3 presents the results of running the algorithm on eleven different RAID-6 erasure codes for 8-disk systems (six data, two parity). The first two codes are RDP and Even-Odd, for which I/O minimization results exist already [23, 13]. The next three are "Minimal Density" that best fit 8-disk systems (Blaum-Roth [2], Liberation [17] and Liber8tion [18]), and the last six are Cauchy Reed-Solomon codes where the variable w, which specifies the number of bits stored per disk, varies from three to eight [3]. For each code, we calculated the average number of bits required for recovery when one data disk fails, plotted as a percentage of the number of bits that are required when matrix inversion is used to decode.

The results show that the Miminal Density codes require fewer bits than RDP and Even-Odd, with the quirky Liber8tion code requiring the fewest bits of all codes.

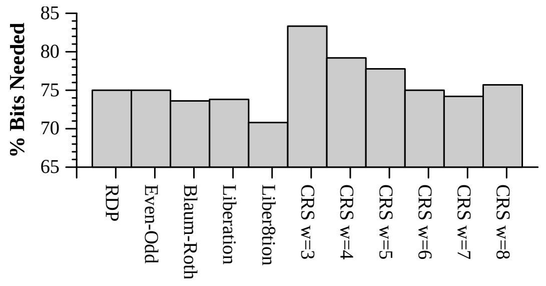

Figure 3: Minimum bits needed for recovering from single failures in RAID-6 codes.

4 An I/O Efficient Recovery Code

Our search for codes which exhibit both low recovery I/O and high fault tolerance led us to consider GRID codes [15] as a suitable candidate. In a GRID code (Figure 4), the disks form a logical grid with each dimension being encoded using (potentially) different schemes. The GRID code allows us to use recovery I/O efficient Weaver codes in conjunction with fault tolerant STAR codes [14], thereby enabling us to capture the desirable properties of both. Weaver codes are parameterized by W(k, t) in which k is the in-degree to a parity symbol and t (fault tolerance) is the in-degree to a data symbol. The fixed in-degree limits the recovery I/O regardless of the stripe size (number of disks). This differs from all systematic erasure codes. Failures within Weaver's fault tolerance in any column are recovered entirely by the Weaver code and benefit from Weaver's efficient recovery I/O.

With a Weaver code, recovery from a single disk failure can be done in two ways. The naive way of recovering from a failure entails accessing the t connected parity disks. But when $k < t$, one can also recover a failed disk by accessing any one of the t parity symbols and then using its connected k data symbols to recover the failed data symbol. Therefore, the cost to recover the failed data symbol is k + 1 I/Os. Parity symbols are recovered using their own k connected disks. Thus, recovery of an entire encoded block takes $(k+1)r + qk = r(t+k+1)$ I/Os (since $rt = qk$), where r and q are the number of data and parity symbols per disk respectively. In some cases, Weaver codes can also recover data and parity from the same disk. This does not reduce the number of I/Os, i.e. block transfers, but does benefit MAID systems in that fewer disks must be spun up.

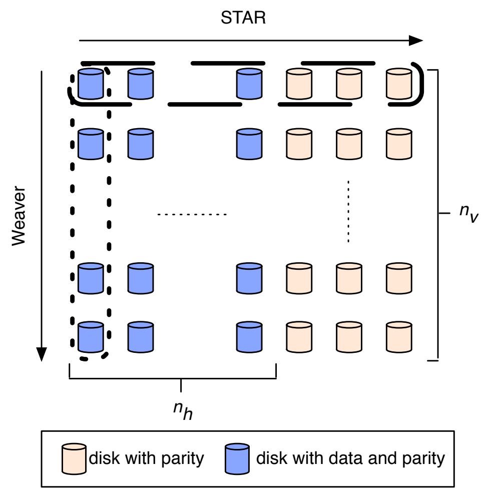

Figure 4: The GRID/Weaver code. The dashed boxes indicate the two separate code dimensions.

	I/Os for recovery	# disks accessed	Storage efficiency	Fault tolerance
GRID(STAR, W(2,2))	4	3	31.25%	11
GRID(STAR, W(3,3))	6	3	31.25%	15
GRID(STAR, W(2,4))	7	4	20.8%	19

Table 1: Performance of GRID/Weaver codes.

Examining the GRID/Weaver construction reveals that the codes use very few I/Os in recovery relative to their fault tolerance and access even fewer disks (Table 1). We experiment with the most efficient (minimum distance separable–MDS) instances of Weaver codes: W(2,2), W(3,3), and W(2,4). We combine this with a STAR code with 5 data disks and 3 parity disks. The GRID(STAR, W(2,2)) tolerates 11 failures and recovers a lost encoded block (data and parity) using 4 I/Os from three disks, in one case accessing data and parity from the same disk. The number of disks that needs to be accessed remains small as fault tolerance increases. Storage overheads for these codes are substantial, but are reasonable given the high fault tolerance and their intended use in archival applications.

5 Discussion and Open Problems

We have corroborated our conjecture that for all XOR-based erasure codes, there is a fundamental tradeoff between recovery I/O and storage overhead at a given fault tolerance. We know the extrema in this tradeoff. Replication has maximum storage overhead and recovers a data block in a single I/O. Minimum distance separable codes provide maximum storage efficiency and the algorithm we present for minimizing recovery I/O gives optimal recovery schedules. We evaluated optimal recovery for the most prevalent XOR-based erasure codes. In between these extrema, lie codes that increase storage overhead and reduce recovery I/O. We demonstrate meaningful intermediate points in the GRID/Weaver code.

It remains an open problem to formalize the tradeoff between storage efficiency and recovery I/O and construct codes that are recovery optimal. We are pursuing this problem both analytically and through automatic erasure code generation. At present, we are conducting a programmatic search of feasible generator matrices and their optimal recovery I/O schedules to find the codes with minimum I/O requirements. The exponential growth of possible codes as a function of matrix size means that we need to develop methods to prune the search both in matrix generation and in the finding of optimal recovery schedules. However, exploring the space for reasonable sized systems, up to one hundred disks, seems within reach.

References

[1] E. Anderson, X. Li, A. Merchant, M. A. Shah, K. Smathers, J. Tucek, M. Uysal, and J. J. Wylie. Efficient eventual consistency in Pahoehoe, an erasure-coded key-blob archive. *Dependable Systems and Networks*, 2010.

[2] M. Blaum and R. M. Roth. On lowest-density MDS codes. *IEEE Trans. on Information Theory*, 45:46–59, 1999.

[3] J. Blomer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, and D. Zuckerman. An XOR-based erasure-resilient coding scheme. Technical Report TR-95-048, International Computer Science Institute, August 1995.

[4] B. Chen, R. Curtmola, G. Ateniese, and R. Burns. Remote data checking for network coding-based distributed storage systems. In *Cloud Computing Security Workshop*, 2010.

[5] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran. Network coding for distributed storage systems. *IEEE Trans. Inf. Theor.*, 56(9):4539–4551, September 2010.

[6] A. L. Drapeau et al. RAID-II: a high-bandwidth network file server. In *International Symposium on Computer architecture*, 1994.

[7] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *USENIX OSDI*, pages 1–7, 2010.

[8] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *ACM SOSP*, 2003.

[9] K. M. Greenan, X. Li, and J. J. Wylie. Flat xor-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs. *Mass Storage Systems and Technologies*, 2010.

[10] J. L. Hafner. Weaver codes:highly fault tolerant erasure codes for storage systems. In *USENIX FAST*, 2005.

[11] J. L. Hafner, V. Deenadhayalan, K. K. Rao, and J. A. Tomlin. Matrix methods for lost data reconstruction in erasure codes. In *USENIX FAST*, 2005.

[12] R. Y. Hou, J. Menon, and Y. N. Patt. Balancing I/O response time and disk rebuild time in a RAID5 disk array. In *Hawaii International Conference on System Sciences*, 1993.

[13] C. Huang, M. Chen, and J. Li. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *Network Computing and Applications*, 2007.

[14] C. Huang and L. Xu. Star : An efficient coding scheme for correcting triple storage node failures. *IEEE Transactions on Computers*, 57:889–901, 2008.

[15] M. Li, J. Shu, and W. Zheng. Grid codes: Strip-based erasure codes with high fault tolerance for storage systems. *ACM Transactions on Storage*, 4(4):15:1–15:22, 2009.

[16] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software—Practice & Experience*, 27(9):995–1012, 1997.

[17] J. S. Plank. The RAID-6 Liberation codes. In *USENIX FAST*, 2008.

[18] J. S. Plank. The RAID-6 Liber8Tion code. *Int. J. High Perform. Comput. Appl.*, 23:242–251, August 2009.

[19] M. Sivathanu, V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Improving storage system availability with D-GRAID. In *USENIX FAST*, 2004.

[20] M. W. Storer, K. Greenan, E. L. Miller, and K. Voruganti. Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage. In *USENIX FAST*, 2008.

[21] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song. PRO: a popularity-based multi-threaded reconstruction optimization for RAID-structured storage systems. In *USENIX FAST*, 2007.

[22] Z. Wang, A. G. Dimakis, and J. Bruck. Rebuilding for array codes in distributed storage systems. *CoRR*, abs/1009.3291, 2010.

[23] L. Xiang, Y. Xu, J. C. S. Lui, and Q. Chang. Optimal recovery of single disk failure in RDP code storage systems. In *ACM SIGMETRICS*, 2010.