Fault Tolerance of Tornado Codes for Archival Storage

Matthew Woitaszek and Henry M. Tufo
University of Colorado, Boulder
matthew.woitaszek@colorado.edu

Abstract

This paper examines a class of low density parity check (LDPC) erasure codes called Tornado Codes for applications in archival storage systems. The fault tolerance of Tornado Code graphs is analyzed and it is shown that it is possible to identify and mitigate worst-case failure scenarios in small (96 node) graphs through use of simulations to find and eliminate critical node sets that can cause Tornado Codes to fail even when almost all blocks are present. The graph construction procedure resulting from the preceding analysis is then used to construct a 96-device Tornado Code storage system with capacity overhead equivalent to RAID 10 that tolerates any 4 device failures. This system is demonstrated to be superior to other parity-based RAID systems. Finally, it is described how a geographically distributed data stewarding system can be enhanced by using cooperatively selected Tornado Code graphs to obtain fault tolerance exceeding that of its constituent storage sites or site replication strategies.

1. Introduction

Archival storage, an essential infrastructure component required by the high-performance computing community, has traditionally been provided by large monolithic tape systems. Magnetic tape is a well established archival storage medium, but tape systems have limited read and write throughput, require tape retrieval queue time, and risk storing all of the important data in one place. The possibility of federated systems that geographically distribute data using disk and tape is an alternative architecture enabled by new storage technologies. For example, massive arrays of idle disks (MAID) have been proposed as a low-power replacement for magnetic tape as a backing store [3]. Grid technologies, such as the Storage Resource Broker (SRB), have also been used to construct distributed mass storage systems using disks for availability and performance while relying on tape to provide redundant backup copies [14]. These developments are particularly relevant to distributed data stewarding initiatives.

The goal of distributed data stewarding is to provide permanence to data that has been deemed to be irreplaceable. Several US agencies and universities are examining the role of digital libraries, mass storage, and data grid technology to construct a nationwide government data archive between existing mass storage systems for use by the National Archives and Records Administration [10]. Other data grid initiatives, such as in the physical sciences, distribute subsets of data collections to facilitate analysis by regional groups [2].

As data preservation is a primary requirement of archival storage, an archival storage system must use a fault tolerance mechanism to reliably store data despite physical device failures. Existing systems, ranging from single-site mass storage systems to data grids, generally use parity and replication for fault tolerance: every tape is mirrored, every disk is protected by membership in a RAID array, and the contents of a single mass storage system are replicated among a community. While replication is a simple approach, it provides much less fault tolerance than systems using erasure coding [15].

We propose constructing an archival storage system using Tornado Codes as the erasure correcting mechanism. Tornado Codes are based on parity calculations similar to those used in RAID but on a larger scale. Instead of a single parity check disk for a small number of disks, a cascaded graph structure describes the generation of a set of parity nodes from data nodes. When data is encoded, an equal amount of parity information is generated and stored, so an encoded file is twice the size of the original file. The original data can then be reconstructed given a subset of the encoded file. In preparation for constructing an archival filesystem using Tornado Codes, we performed an empirical analysis to compare the fault tolerance of Tornado Codes to other parity-based erasure coding schemes. For our analysis, we focus on fault tolerance in a potentially geographically distributed archival storage environment irrespective of a specific backing store selection.

In addition to single-site mass storage systems, Tornado Codes may also be used to increase fault tolerance among distributed archival storage where data is replicated among sites. By choosing complementary Tornado Code graphs for each site, even reconstruction failures at all sites may be recoverable by exchanging a small number of blocks. Rather than a revolutionary change requiring new storage paradigms, Tornado Coded storage fits well within the existing taxonomy of archival storage and is useful for single-site mass storage and multi-site data stewarding systems.
The remainder of this paper is organized as follows. Section 2 describes Tornado Codes and their prior application to multicast and storage. Section 3 examines the fault tolerance of Tornado Code graphs, identifying worst case failure scenarios and describing how graph adjustments can increase fault tolerance. Section 4 compares the fault tolerance provided by Tornado Code graphs to alternate approaches, and the final sections describe how complementary Tornado Code graphs can be used to construct a geographically distributed archival storage system.

2. Background and Related Work

Tornado Codes were originally described by Luby as a mechanism to perform reliable multicast [8]. The basic unit of a Tornado Code is a bipartite low density parity check (LDPC) graph (see Figure 1). A series of data nodes represents the original data to be stored and check nodes are derived to provide redundancy. Graph edges describe which data blocks are used when calculating parity using XOR. The codes are described as low density because they contain fewer edges than a fully connected graph.

A Tornado Code consists of a series of cascaded irregular bipartite graphs connecting data nodes to check nodes (see Figure 2). The LDPC coding scheme is repeated at each level, with the left nodes being used to calculate right parity check nodes. Reconstruction is performed in reverse: if any right node has exactly one missing left node, the missing left node can be reconstructed. Parameters involved in the construction of a Tornado Code include the number of nodes, the number of levels, and the left and right edge distributions between each level.

Luby first presented Tornado Codes as a mechanism for reliable and efficient multicast file distribution on the Internet called Digital Fountain [1]. Intended to provide a mechanism for popular sites to serve content to large audiences, Digital Fountain encodes a file using Tornado Codes and then broadcasts blocks in a non-overlapping order from multiple servers. Clients receive blocks from the broadcasting servers until a sufficient number of packets have been received and the original file can be recreated.

2.1. Tornado Codes in Storage Systems

In a peer-to-peer or distributed filesystem, files are segmented into blocks and stored on a large number of geographically separated clients. Erasure codes have been shown to provide greater fault tolerance than replication in distributed filesystems [15], and several groups have examined the application of Tornado Codes and other LDPC codes in this context. Relevant related work includes the development of alternatives to Tornado Codes, an examination of real LDPC codes for finite graphs, and a prototype implementation of a distributed filesystem using Tornado Codes within the Oceanstore framework.

Lincoln Erasure Codes (LEC) were presented as a higher-throughput and more fault tolerant alternative to Tornado Codes [4]. The LEC algorithm is similar to Tornado Codes but utilizes a different distribution of edges between data and check nodes. The authors focused on the automated generation and evaluation of suitable LEC graphs and report that LEC provides substantially higher throughput than Luby’s codes with less block failure probability. Although LEC appears to be a feasible alternative to Luby’s patented Tornado Codes, we chose to examine Tornado Codes for our current work. As the software developed for our work can utilize any LDPC graph, evaluation of LEC graphs in future work is possible.

Recognizing that the original work on LDPC codes focused on the “collective and asymptotic” performance of the coding algorithms, Plank and Thomason performed one of the first analyses of realized codes for finite sized graphs [12]. Plank examined the relative performance of graphs within several families, described how well published algorithms produced usable codes, and compared the LDPC codes to Reed-Solomon coding. One of Plank’s most important contributions is a description
of how LDPC codes work with small numbers of nodes: poorly. Plank concludes that LDPC codes demonstrate their least favorable overhead for graphs containing between 10 and 100 nodes.

Our analysis is complimentary to Plank’s work. Plank focused on graph efficiency by identifying the fewest number of nodes required to successfully reconstruct data. Our simulation methodology identifies the fewest number of nodes required to lose data, an important consideration for a storage system claiming fault tolerance. The simulation reports the probability of successful graph reconstruction given an a priori number of offline nodes, a metric which can be used to quantify the fault tolerance and calculate reliability given device failure characteristics.

The most substantial work involving Tornado Codes in distributed storage was performed as part of the Typhoon project at the University of California, Berkeley [5], which prototyped the use of Tornado Codes in Oceanstore [7] for distributed archival storage. Typhoon generates Tornado Code graphs on a just-in-time, per-file basis, and stores the resulting blocks using the pre-existing Oceanstore framework. The authors found that Tornado Codes encode and decode files in substantially less time than Reed-Solomon codes. While the empirical failure rate was not determined, the authors note that files become inaccessible at 40-45% block unavailability. The most important component of the authors’ work is a concise algorithm for constructing Luby’s Tornado Codes that we used as a basis for generating our graphs.

2.2. Archival Storage Interfaces

Rather than providing block-based access to data such as in a traditional filesystem, archival storage systems typically work at the granularity of files or collections. This access method makes Tornado Codes a good technique for archival storage as opposed to random access storage. Mirroring and RAID remain good choices for systems supporting block-based access because of the limited number of devices involved in any read or write operation. Block-level access presents challenges for any large-scale LDPC coding scheme because during an in-place update, modifying a single block may require a large number of parity blocks to be updated through the cascaded graph stages. Thus, using a Tornado Code as an alternate to RAID on a high-performance parallel filesystem is not appropriate.

While a block-based filesystem allows a single block in a file to be updated in place, archival systems function using a transactional interface where complete files or objects are uploaded or downloaded. Because the size of the object to be stored is known at the time the storage is requested, Tornado Codes can be used to encode data without the complexities associated with incremental block-based updates. This makes Tornado Codes a technically feasible erasure coding scheme for archival storage.

In addition to networked distributed storage, single-site MAID systems also could benefit from LDPC erasure coding techniques for both fault tolerance and performance. The initial work with MAID systems used a block-level interface and examined methods to place data and metadata blocks to minimize the system’s power consumption [3]. COPAN Systems currently constructs MAID systems using large arrays of SATA disks. Their current product uses “a highly unique RAID that maximizes data availability through granular power management and still provides protection while powering on and off disks” [9]. LDPC codes can survive more failed devices than mirroring or RAID parity. In addition, distributing data among devices with LDPC coding may provide opportunities to optimize the block retrieval process, minimizing the number of devices accessed to retrieve data. Thus, the combination of Tornado Codes and MAID may be capable of producing a highly reliable and power efficient archival storage system.

3. Design

To evaluate the fault tolerance of Tornado Codes, we designed and implemented a Tornado Code graph generator and an automated testing system. The testing system stores graphs in the standardized GraphML format to simplify graph visualization and editing. For example, the testing suite can render failed graphs highlighting unrecoverable nodes and check node dependencies related to the graph failure.

For each graph, two metrics are of interest: the worst case failure scenario and the fraction of reconstruction failures for a given number of missing nodes. The worst case failure describes the minimum number of missing nodes that makes a LDPC graph unrecoverable. Translated into a storage system, it describes the minimum number of failed disk drives that may cause data loss. For example, consider a traditional high performance storage system containing 10 RAID5 LUNs. The worst case failure scenario is two as the failure of two disks in a single LUN cause data loss.

The second metric is the fraction of reconstruction failures for a given number of missing nodes. In the example RAID5 system with 10 LUNs, the system could support the loss of ten drives as long as exactly one drive fails in each LUN. In the case where 11 disks fail, data loss is guaranteed because one LUN must have lost at least 2 disks. Between the worst case failure scenario and the guaranteed failure scenario is a range where failure is probable depending on how many and which specific disks fail. To describe a graph’s behavior in the transition regime, we calculate the fraction of failed reconstructions.
given a specific number of offline devices. By using a large number of test cases, this produces the probability of graph failure given a number of offline devices.

In our implementation, we used 96-node graphs. Even though Plank’s results show that this is near the region where LDPC codes have unfavorably high overhead [12], it is an appropriate lower bound for filesystem construction purposes. Assuming an average 50% overhead, a file can be reconstructed by accessing 72 devices, a reasonable number of devices to retain the ability to explicitly manage device status. In addition, in a MAID system with 2000 disks, this allows several stripes to be accessed concurrently while limiting the number of drives online to a small percentage. Using larger device counts in a coded stripe may be appropriate in larger systems, but using fewer nodes is not feasible. The 96 nodes used in this analysis balances computational tractability with implementation feasibility.

We detect worst case failure scenarios using a full combinatorial examination of lost nodes, starting with (96 choose 1 lost block) through (96 choose 6). This is computationally tractable due not to the limited size of the graph but the fact that we were unable to find a graph that could tolerate more than five failed nodes in its worst case scenario. Thus, it was simple to identify the worst case scenario for every graph, and in our environment the test set requires only 21 CPU hours to complete for each graph.

The second test suite calculates the fraction of failed reconstructions for a large number of test cases. The combinatorial expansion between (96 choose 1) and (96 choose 48) is not computationally tractable, so we test a subset of random failure cases for each number of lost devices. The number of test cases for each data point ranges from over 10 million for 5 devices offline to over 34 million for 48 devices offline, with the larger subsets capped to produce consistent simulation times. The test suite contains exactly 962,144,153 test cases and requires about 34 CPU days to execute for a single graph. To verify the correct operation of the simulator, we created a 96-node mirrored system using our graph generation tool and verified that the sampled results were equal to the theoretical values produced by Equation (1), where \( n \) is the number of mirrors in the array (half the number of disks):

\[
P(\text{fail} | k \text{ drives offline}) = \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n}{j} \binom{n-1}{k-2j} 2^{\frac{j(2j-2)}{k}} \]  

(1)

For this mirrored graph, our simulator and test suite produced values equal to the theoretical values to at least 9 significant digits. Therefore, we believe that the sampling system can produce graph failure probabilities that at least approximate a complete search. We use this simulator to calculate the performance of specific random graphs for which theoretical values are not available.

3.1. Generating Tornado Codes

We used a combination of the procedures described by Luby [8] and the Typhoon implementation [5] to generate Tornado Code LDPC graphs. We retained Luby’s preference for working with degrees of edges instead of degrees of nodes, but added an intermediate processing step to adjust the desired degree distributions to produce the required number of nodes. When working with such small graphs, we frequently encountered situations where the distribution suggested 5 edges of degree 6, which is meaningless because by definition an edge of degree 6 must be connected to a node with 6 edges. In the latter stages of the graphs, most of the edge counts would be less than the edge degree and produce no nodes. Rather than perform an iterative adjustment to simply add missing nodes, we implemented a numeric solver to find a constant multiplier for the edge distribution that produced the correct number of nodes.

We used Typhoon’s suggested treatment of the final Tornado Code levels. In this arrangement, the last two stages of the graph share the same set of left nodes. When implemented, this makes a graph with two final stages containing 4 nodes each appear to have one final stage with 8 nodes, but each right node set of 4 is calculated independently using the whole set of 8 left nodes. This method seemed appropriate because it was difficult to construct a meaningful edge distribution between two levels of size 4. The resulting graph constructor was able to produce Tornado Code graphs as small as 32 total nodes.

3.2. Initial Graph Failure Experiences

After implementing the Tornado Code generator, we performed initial tests on a half-dozen graphs and discovered that some of the graphs contained obvious defects. Most egregious was the case where a set of left nodes shared the same right nodes. Written in the form “left node [ right nodes ]”, one graph contained:

- 17 [ 48, 57 ]
- 22 [ 48, 57 ]

In this case, two left nodes use exactly the same two right nodes for redundancy. If both data nodes in the set (17, 22) are lost, even the presence of all the other nodes in the graph cannot help reconstruct either lost data node, and the graph’s worst case failure scenario is two. These trivial cases are easily detected and corrected. However, the same situation arises for larger sets. For example:
This set of three left nodes fails because each left node uses right nodes that are members of a small closed set. This is one typical way that Tornado Codes fail. Sets of left nodes rely on closed sets of right nodes, but right nodes are only useful if a sufficient number of left nodes are available because the reconstruction algorithm can only recover data from a right node that has lost exactly one left node.

The fewest number of nodes required to reconstruct a Tornado Code is the data nodes themselves. However, having no data nodes and all check nodes is useless. Thus, reconstruction is only possible given a subset of nodes that includes the nodes in critical left node sets. Small left node failure sets limit the fault tolerance. The problem with random graph generation is that bad edge connections occasionally happen. Rather than high fault tolerance, badly constructed Tornado Codes can fail if a small set of critical devices fail. Our worst case failure analysis detects these critical node sets and attempts to mitigate them.

3.3. Adjusting Graphs for Increased Fault Tolerance

We employed two strategies to produce Tornado Code graphs with more favorable fault tolerance characteristics. The first approach involved structural defect detection as part of the graph generation procedure. As a graph is produced, it is subjected to several tests to detect obvious problems such as two- and three-node overlapping sets, and graphs that fail are discarded. As the number of nodes involved in failures increases, detecting problems becomes more difficult and the likelihood of generating a passable graph diminishes. We did not attempt to manufacture the perfect LDPC graph because Tornado Code graphs are based on random construction by design. Instead, we switched to another strategy of detecting failure patterns with the testing suite and using the failure data to alter the graph. This approach has worked quite well in practice, producing slightly larger critical set sizes for worst case failure.

For example, we first tested one prototype graph using every (96 choose 4) failure case. Exactly two out of 3,321,960 test cases resulted in reconstruction failure involving the following sets of left nodes:

\[ 6 \ [48, 51, 57] \]
\[ 28 \ [57, 66, 68] \]
\[ 42 \ [48, 51, 66, 68] \]

In this example, all loss combinations of three nodes are tolerated, and exactly two sets of four losses result in data failure. Because the number of loss cases is quite small, it was possible to perform a manual tweak of the graph to remove the failure case.

To perform the adjustment, we first identify critical left nodes that were involved in the most failure sets. In the example above, node 23 is involved in both failure sets and is the target node for adjustment. Each failure set was influenced by a closed set of right nodes, so the target left node’s right nodes are compared to the other right nodes involved in the failure set. For the target left node, we find the right node with the highest failure rate and then change the connectivity of the target left node to include a different right node that was not involved in the failures. This opens the closed set that caused the failure and removes the failure set provided that the substitution did not tie one failure set to another.

After the adjustment has been completed, the adjusted graph is re-tested. In this example, the adjusted graph had no failures with the loss of four devices, and 14 losses out of 61,124,064 when considering 5 lost devices. For this graph, the adjustment worked well, but the success of the algorithm is dependent on the graph. A successful graph tweak requires that the graph have more candidate right node replacements than left node failure sets requiring adjustments and is ultimately related to the degree of the graph. As the average degree of our graphs was 3.6, we were generally able to tweak each graph to tolerate 4 failures, but when attempting to increase to 5 failures insufficient candidates for replacement were available. Forcing an adjustment with bad replacement nodes corrects the target set but creates new failure sets.

4. Results

To find Tornado Code graphs for implementation in a storage system, we examined the fault tolerance characteristics of several families of graphs and determined the effect of our fault detection and mitigation strategies. Our results show that Tornado Code graphs, as constructed by our graph generation system with the defect detection in place, clearly provide better fault tolerance than existing RAID systems and unadjusted Tornado Code graphs. In addition, the Tornado Code graphs outperform regular and random single-level and cascaded non-Tornado LDPC graphs.

We avoided the temptation to search for the Holy Grail of Tornado Code graphs. Instead of subjecting thousands of potential graphs to thousands of tests, we subjected about a dozen potential graphs to millions of tests. Graphs that exhibited unacceptable recoverability from the beginning were discarded. The remaining graphs, all of which experienced first failures at 4 lost nodes, were analyzed and adjusted to experience first failures at 5 lost nodes. These graphs are candidates for implementation in an archival filesystem.
Because each graph has slightly different recovery properties, we present results for a subset of the three best graphs individually labeled by number. For each erasure coding graph, we present the number of nodes involved in the first failure, a plot of the fraction of test cases failing reconstruction by number of offline nodes, and the average number of nodes that was capable of reconstructing the data. The number of nodes involved in the first failure is the most important factor increasing reliability, as the probability of losing an increasing number of drives at the same time diminishes rapidly. Tolerating the failure of more simultaneous failures provides greatly improved reliability.

The plots are useful as a comparative measure of each graph’s relative performance under increasing loss scenarios. This is useful to identify a starting retrieval count for reconstruction operations in an implementation. The final metric, the average number of nodes capable of reconstructing the data, describes the graph’s transition point from success to failure. It is important to note that the reported average number of nodes required to reconstruct is not the overhead as described in the literature. To determine the overhead of a graph, a testing system would start with a certain number of online nodes and retrieve nodes until the graph can be reconstructed. In our testing system, the number of online nodes is set in advance and the test case is recorded as passing or failing reconstruction with that node count. This is not overhead because it does not measure the minimum quantity of data required to reconstruct. While this does not describe overhead, the advantage is that the testing sets are independent and may be used to calculate reliability during subsequent analysis.

4.1. RAID and Best Tornado Code Graphs

Our results confirm that storage using Tornado Code error correction has greater fault tolerance than mirrored systems, even though both systems have the same storage overhead of 50% (see Figure 3 and Table 1). The mirrored system can fail when as few as two devices fail, but the Tornado Coded system can support the loss of any four devices. It can only lose data when five devices fail, and even then the probability of data loss is extremely low: less than 14 cases out of 61,124,064.
Figure 5. Fraction reconstruction failure by number of missing nodes in a 96-node storage system for selected Tornado and alternate graphs

Table 3. First failure and average number of nodes capable of reconstructing data in 96-node storage systems for selected Tornado and alternate graphs

<table>
<thead>
<tr>
<th>System</th>
<th>First Failure</th>
<th>Average to Reconstruct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular – Degree = 4</td>
<td>4</td>
<td>77.49 (1.61)</td>
</tr>
<tr>
<td>Regular – Degree = 11</td>
<td>4</td>
<td>78.61 (1.63)</td>
</tr>
<tr>
<td>Altered Tornado (dist. doubled)</td>
<td>5</td>
<td>77.41 (1.61)</td>
</tr>
<tr>
<td>Altered Tornado (dist. shifted)</td>
<td>5</td>
<td>75.58 (1.57)</td>
</tr>
<tr>
<td>Tornado Graph 3 (best)</td>
<td>5</td>
<td>73.77 (1.53)</td>
</tr>
</tbody>
</table>

The Tornado Code systems also perform substantially better than RAID systems using the same number of disks. This comparison is skewed because RAID systems use fewer disks for parity. The RAID systems are configured as 8 drawers with 12 disks per drawer. Thus, the RAID5 system has 8 parity disks, the RAID6 system 16 parity disks, and the mirrored and Tornado systems 48 parity disks. The RAID systems will present greater capacity to the end-user through the use of fewer disks for parity storage, but the parity disks provide selection flexibility for Tornado Coded storage.

4.2. Adjusted Tornado Code Graphs

The structural defect detection and graph adjustment procedures were successful in producing Tornado Code graphs with increased fault tolerance (see Figure 4 and Table 2). The worst initial prototype graphs without any form of defect detection failed at two nodes, but the introduction of defect detection increased the first failure for new graphs to four nodes. The feedback-based graph adjustment procedure was able to increase the fault tolerance of the graphs by one more node, and the best graphs demonstrated first failure with 5 lost nodes.

Figure 6. Fraction reconstruction failure by number of missing nodes in a 96-node storage system for fixed-degree cascaded random graphs and Tornado Code graphs

Table 4. First failure and average number of nodes capable of reconstructing data in 96-node storage systems for fixed-degree cascaded random graphs and Tornado Code graphs

<table>
<thead>
<tr>
<th>System</th>
<th>First Failure</th>
<th>Average to Reconstruct</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cascaded – Degree = 6</td>
<td>5</td>
<td>80.39 (1.67)</td>
</tr>
<tr>
<td>Cascaded – Degree = 4</td>
<td>4</td>
<td>76.60 (1.59)</td>
</tr>
<tr>
<td>Cascaded – Degree = 3</td>
<td>4</td>
<td>74.00 (1.54)</td>
</tr>
<tr>
<td>Tornado Graph 3 (best)</td>
<td>5</td>
<td>73.77 (1.53)</td>
</tr>
</tbody>
</table>

4.3. Non-Tornado Code Graphs

In addition to the RAID-based and Tornado Code graphs, we also tested other graph distributions (see Figure 5 and Table 3). Regular single-stage graphs, such as those of degree 4 and 11, performed poorly. We also attempted several alterations of Tornado Code graphs. For example, these adjustments doubled the degree distribution or shifted the degree distribution +1 edge. Altering Tornado Code graphs by increasing the connectivity generally increased the first failure but with the penalty of an earlier average failure point.

To examine this further, we tested a series of randomly generated fixed-degree cascading LDPC graphs. These graphs have the same number of stages as Tornado Codes and use a random edge distribution, but instead of the varying Tornado Code degree distribution the degree was fixed. The reconstruction profile of a regular graph with degree 3 almost matched that of the best Tornado Code graph, which has a degree of 3.6 (see Figure 6 and Table 4), but experienced its first failure earlier. Increasing the connectivity initially increases the tolerance to failure because left nodes can be reconstructed with the
assistance of additional right nodes. However, with too much connectivity, right nodes become incapable of assisting with reconstruction because a right node is only useful if it has exactly one missing left neighbor.

5. Discussion

For an archival filesystem constructed using Tornado Codes, reliability and reconstruction efficiency are of primary importance. The increased reliability over existing storage solutions makes an archival system using Tornado Codes compelling. Furthermore, the ability to balance block retrieval and reconstruction may allow optimizations to reduce file retrieval time. We plan to examine this in future work.

5.1. Reliability

Reliability is the probability of no failure occurring within a specific time period [13]. The preceding results examined graph fault tolerance in a time-neutral context, describing only the probability of data loss given a specific number of unavailable graph nodes. In order to calculate reliability, a specific hardware mapping with device failure characteristics must be introduced. For comparison purposes, our theoretical storage system contains 96 individually-accessible drives that can be configured using RAID, striping and mirroring, or Tornado Code erasure coding. We assumed an annual failure rate (AFR) of 1%, \( p = 0.01 \), as a conservative estimate of drive failure rates consistent with the literature [6], and then use the device AFR to calculate reliability for the entire system. We neglect the effect of controller failures because our primary concern is data reliability and not the availability of the system with dependencies on hypothetical hardware components and timely replacements.

The probability that \( k \) drives in an \( n \)-drive array have failed is given by the binomial distribution based on the probability of failure of a single drive:

\[
P(k \text{ drives lost}) = (p)^k (1-p)^{n-k} \binom{n}{k}
\]  

(2)

Then, \( P(\text{fail} | k \text{ drives lost}) \) is the experimentally calculated fraction of unrecoverable graphs given a set number of missing nodes. Because the failure cases for each number of offline devices are independent, the probabilities may be summed to determine the probability of failure for the system in Equation (3):

\[
P(\text{fail}) = \sum_{k=0}^{n \text{ drives}} P(\text{fail} | k \text{ drives lost}) P(k \text{ drives lost})
\]  

(3)

Table 5. Theoretical probability of failure (data loss) for 96-disk systems assuming independent disk failures with an annual failure rate \( p = 0.01 \) and no repair

<table>
<thead>
<tr>
<th>System</th>
<th>Data</th>
<th>Parity</th>
<th>( P(\text{fail}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual Disk</td>
<td>96</td>
<td>0</td>
<td>0.01000</td>
</tr>
<tr>
<td>Stripping</td>
<td>96</td>
<td>0</td>
<td>0.61895</td>
</tr>
<tr>
<td>RAID5</td>
<td>88</td>
<td>8</td>
<td>0.04834</td>
</tr>
<tr>
<td>RAID6</td>
<td>80</td>
<td>16</td>
<td>0.00164</td>
</tr>
<tr>
<td>Mirrored</td>
<td>48</td>
<td>48</td>
<td>0.00479</td>
</tr>
<tr>
<td>Tornado Graph 1</td>
<td>48</td>
<td>48</td>
<td>1.34E-09</td>
</tr>
<tr>
<td>Tornado Graph 2</td>
<td>48</td>
<td>48</td>
<td>5.947E-10</td>
</tr>
<tr>
<td>Tornado Graph 3</td>
<td>48</td>
<td>48</td>
<td>5.857E-10</td>
</tr>
</tbody>
</table>

System failure rates derived from the experimental graph failure profiles and our analysis confirm that striping provides reduced reliability, RAID5, RAID6, and mirroring provide additional reliability, and Tornado Codes provide substantially more protection from device failures (see Table 5). Because of the reduced probability of an increasing number of simultaneous failed disks, the reliability of the entire system is dominated by the worst case failures. For example, mirroring and RAID may fail when 3 devices are offline, and \( P(\text{exactly 3 disks fail}) = 0.056 \). The first failure with the Tornado Code graphs occurs when 5 disks fail, but \( P(\text{exactly 5 disks fail}) = 0.0024 \). When summing the independent component probabilities of failure given each device count from zero to the total number of disks, the first failure provides the greatest contribution to the system failure rate. By tolerating the failure of a larger number of devices, Tornado Codes provide substantially greater reliability.

5.2. Reconstruction Efficiency

Reliability and fault tolerance emphasize the minimum number of missing blocks that can cause critical data loss, but in normal operating circumstances the opposite situation, represented by the reconstruction overhead, is of interest: what is the average minimum number of blocks that can be used to reconstruct a file? We chose to address worst case fault tolerance first because reliability is a critical factor in the adoption of any storage system, and previous work on LDPC codes in storage suggested overheads less than 1.2 [11],[12]. However, in a functioning archival system – especially one based on MAID where disks must be powered on – the minimum set of blocks may not always be the best set to retrieve. Instead, retrieving an optimal set of blocks and reconstructing the data may be preferable.

To describe the reconstruction efficiency of the encoding method, we determined the minimum number of nodes that provide a 50% probability of being able to reconstruct the stripe and then calculate overhead from that number of nodes (see Table 6). From the results,
retrieving 62/96 blocks will be sufficient to immediately reconstruct the data half of the time, and half the time additional nodes will need to be retrieved. This overhead value of 1.29 is larger than the values in the literature because our node count for 50% success includes cases where a fewer number of nodes would have been able to reconstruct the data. In a filesystem implementation, the unnecessary nodes would not be retrieved.

5.3. Multi-graph Distributed Archival Storage

Federated archival storage systems rely on a small number of sites with mass storage capabilities to provide capacity and use grid technology for data access and transfer. For example, a digital library may use SRB for metadata collection and place replicas of files among member sites. Each site may store data using its own fault tolerance scheme, such as RAID or replication, but data is replicated among sites for additional reliability.

Just as in the case of individual sites, the use of Tornado Codes can provide substantially more fault tolerance than simple replication within the same storage footprint. We propose constructing federated archival storage systems using replication among sites, just as is done with many data grids at the present time, but with each site using Tornado Codes internally instead of replication. By using complimentary Tornado Code graphs, the distributed systems can achieve fault tolerance in excess of that of the individual member sites.

To explore this further, we simulated a federated storage system with two replicas. All data is replicated between at least two sites, and each site uses a different Tornado Code graph for erasure protection. In the mirrored configuration, both sites use replication to protect data locally, so each unique block has 4 copies throughout the entire system. The other configurations examine combinations of Tornado Codes in pairs. Because of the much larger number of devices, the first failures could not be located using a brute force approach. Instead, we used the previously detected failure cases for the 96-node graphs to construct test cases that examined the situations where graph failure was known to occur.

The results show greatly increased fault tolerance for the multi-graph encoding schemes (see Table 7). The mirrored approach that stores 4 copies of all blocks can fail when 4 devices are lost. The multi-graph system using two instances of the same Tornado Code graph experiences its first failure with the loss of 10 devices as expected because the component graphs show failure with the loss of 5 devices. The complimentary graph approaches have a much higher first failure point ranging from 17 to 19 first failures.

This higher first failure point is due to the nature of failures in Tornado Code graphs. When a Tornado Code graph fails, it is generally due to a small subset of data nodes that could not be reconstructed. Each Tornado Code graph has different data node dependency sets, so when one graph fails the other may not. Failure of both graphs only occurs when both graphs are missing a critical set of left nodes, and the same set must be lost at both sites. By allowing the replicas to exchange the missing data nodes, restoring just one critical data node allows the data graph to be reconstructed even when both graphs cannot independently perform the reconstruction.

6. Future Work

The next step of our plan to construct an archival storage system using Tornado Codes is to examine reconstruction efficiency and retrieval optimizations. In particular, we plan on examining several guided search techniques to minimize the number of devices accessed to reconstruct an encoded stripe, first on a per-stripe basis in a null environment, and then in a system where multiple stripes must be reconstructed at the same time within a stateful environment. We also will evaluate new theoretical work on small LDPC graphs with more optimal overhead for storage applications [11], possibly implementing and testing new graphs to replace our original Tornado Code graphs.

Following initial optimization experiments, we intend to construct a prototype archival storage system. Our preliminary architecture, still under design, is backing-store neutral and will support standardized object storage targets. One important feature of the proposed system is a stripe reliability assurance and user introspection mechanism to proactively monitor the status of distributed encoded stripes and reconstruct missing blocks before a stripe approaches the initial failure point. Our final objective is a working prototype archival storage system.

<table>
<thead>
<tr>
<th>System</th>
<th>Nodes</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tornado 1</td>
<td>62</td>
<td>1.29</td>
</tr>
<tr>
<td>Tornado 2</td>
<td>62</td>
<td>1.29</td>
</tr>
<tr>
<td>Tornado 3</td>
<td>61</td>
<td>1.27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System</th>
<th>First Failure Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mirrored (4 copies)</td>
<td>4</td>
</tr>
<tr>
<td>Tornado 1 + Tornado 1</td>
<td>10</td>
</tr>
<tr>
<td>Tornado 1 + Tornado 2</td>
<td>17</td>
</tr>
<tr>
<td>Tornado 1 + Tornado 3</td>
<td>17</td>
</tr>
<tr>
<td>Tornado 2 + Tornado 3</td>
<td>19</td>
</tr>
</tbody>
</table>

Table 6. Number of nodes required for 50% probability reconstruction success and resulting graph overhead for a 96-node graph

Table 7. First failure detected for a multi-graph archival storage system using two Tornado Code graphs
7. Conclusion

Tornado Codes provide erasure coding with probabilistically successful data reconstruction. As the first step in constructing a distributed archival storage system using Tornado Codes, we quantified “probabilistically successful” by determining the worst case fault tolerance and reliability of data protected by several actual Tornado Code graphs. Our results show that Tornado Codes provide greater fault tolerance than replication with the same space requirement.

Although Tornado Codes can provide substantial fault tolerance, it is important that generated graphs be thoroughly tested before implementation in a storage system. A storage system using Tornado Codes where data loss must be avoided should use precompiled graphs and not random graphs or perform basic worst-case fault detection on new graphs before use. Randomly generated graphs may contain structural defects that dramatically reduce their tolerance to failure.

In the future, we plan on using these profiled Tornado Code graphs to construct a federated archival storage system that distributes data among several mass storage sites. Each site may be based on existing technology, such as object storage, or take advantage of MAID-based backing stores for reduced power consumption. By using Tornado Codes, this storage system will provide highly reliable storage space while functioning within existing data grid infrastructure.

Acknowledgements

We would like to Dirk Grunwald for his suggestion of using Tornado Codes to construct a filesystem for MAID storage systems, as well as Jason Cope, Sean McCreary, and Michael Oberg for their valuable feedback. Computer time in excess of 6 CPU years was provided by NSF ARI Grant #CDA-9601817, NASA AIST grant #NAG2-1646, DOE SciDAC grant #DE-FG02-04ER63870, NSF sponsorship of the National Center for Atmospheric Research, and a grant from the IBM Shared University Research (SUR) program.

References


