Tornado Codes for MAID Archival Storage

Matthew Woitaszek
University of Colorado, Boulder
matthew.woitaszek@colorado.edu

Henry M. Tufo
University of Colorado, Boulder
tufo@cs.colorado.edu

Abstract

This paper examines the application of Tornado Codes, a class of low density parity check (LDPC) erasure codes, to archival storage systems based on massive arrays of idle disks (MAID). We present a log-structured extent-based archival file system based on Tornado Coded stripe storage. The file system is combined with a MAID simulator to emulate the behavior of a large-scale storage system with the goal of employing Tornado Codes to provide fault tolerance and performance in a power-constrained environment. The effect of power conservation constraints on system throughput is examined, and a policy of placing multiple data nodes on a single device is shown to increase read throughput at the cost of a measurable, but negligible, decrease in fault tolerance. Finally, a system prototype is implemented on a 100 TB Lustre storage cluster, providing GridFTP accessible storage with higher reliability and availability than the underlying storage architecture.

1. Introduction

The high performance computing field has a nearly insatiable demand for storage. Performance, availability, reliability, cost, capacity, and scalability are common metrics used to evaluate storage, but the environment ultimately dictates the choice of technology. Though all storage systems strive for reliability, it is of paramount importance for archival storage. Intended to serve as permanent repositories of data, archival storage systems employ data redundancy techniques, such as replication, erasure correcting codes, or error correcting codes, to reduce the risk of data loss.

The emergence of new technologies has renewed interest in exploring alternative architectures for archival storage. For example, MAID (Massive Arrays of Idle Disks) has been proposed as a low-power and lower-latency replacement for magnetic tape as a backing store [3]. Grid technologies have been used to construct distributed storage systems for data preservation [6] that include both disk and tape [8]. Tornado Codes and other erasure codes have been shown to provide high levels of fault tolerance. This confluence of new technologies holds the potential of radically altering our concept of archival storage by providing solutions that exhibit an unprecedented combination of fault tolerance, performance, availability, low power consumption, and ability to be federated among collaborating institutions.

In our prior work, we examined the fault tolerance of specific Tornado Code graphs, focusing on worst-case data loss scenarios and overall system reliability, and demonstrate that Tornado Codes with iterative decoding provide higher reliability than replication and RAID [10]. Having demonstrated the feasibility of Tornado Codes for storage applications, we now develop and implement the Tornado Coded Archival Storage (TCAS) system. The TCAS file system is then combined with a MAID simulator to emulate a large-scale storage system, and is evaluate using workload traces. Finally, we develop a GridFTP front-end to the TCAS system that allows clients to archive and retrieve files using a standard interface.

2. Background and related work

Tornado Codes were introduced by Luby as a mechanism for reliable multicast [2]. The basic unit of a Tornado Code is an irregular bipartite low density parity check (LDPC) graph. The nodes of the graph store either data or parity, and the randomly paired edges describe which data nodes are combined using XOR to produce parity nodes. A Tornado Code is constructed by cascading several of these bipartite graphs. Tornado Codes may be created with variable quantities of nodes and stages, but it is the specific edge degree distribution that makes a cascaded bipartite LDPC graph possess the fault tolerance characteristics of a Tornado Code.

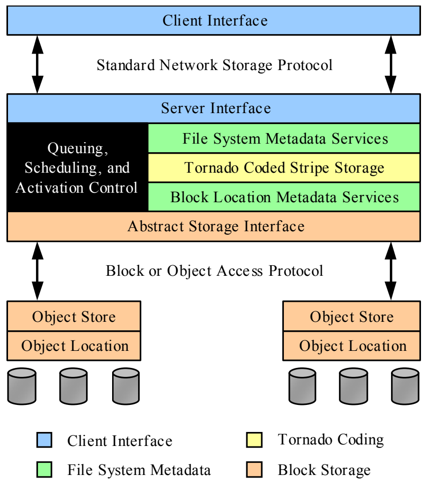
Because erasure codes provide better fault tolerance than replication [9], they have been of particular interest to distributed file system developers. OceanStore originally included an archival storage class using Reed-Solomon codes, but in 1999 the Typhoon project examined using Tornado Codes as a replacement coding method [4]. Typhoon demonstrated that Tornado Codes could encode and decode data in substantially less time than Reed-Solomon codes. More recently, RobuSTore used a derivative of Luby's LT codes to hide latency in a distributed storage system using speculative access [11].

While Tornado Codes and LT codes are increasingly popular coding methods for storage applications, alternative approaches for generating LDPC graphs have been proposed. For example, Plank and Thomason's analysis of specific graphs for several families of codes demonstrated that random LDPC codes work best with large numbers of nodes, and Plank's continuing work describes the construction of optimal codes for small (less than 100 node) graphs [7]. We have chosen Tornado Codes over other codes for the TCAS system because of their simplicity and prior applications in the literature. However, our software works with any graph-based fixed-rate code.

In preparation for constructing a file system using Tornado Codes, we analyzed the fault tolerance of specific 96-node Tornado Code graphs in our prior work [10]. Through simulations, we identified the worst case failure scenarios for several graphs. Unsuitable graphs were discarded, and the remaining graphs demonstrated the ability to withstand the failure of any 4 out of 96 devices using iterative decoding. The ability to tolerate any four lost devices provides fault tolerance greater than RAID and mirroring. A single graph was selected for use with the storage implementation presented here.

3. System design

The TCAS system is intended to serve as large-scale single-site or federated storage system. We envision a system that functions as a data grid appliance similar to a Grid Brick but with larger capacity, configurable power constraints, and higher throughput and lower latency than tape-based systems. Our overall system architecture consists of a Tornado Coded file system, a system controller server running the file system, backing store such as a MAID device, and a user interface based on established data grid standards. A layered component software architecture implements the TCAS system data flow (see Figure 1).



Client Interface
Standard Network Storage Protocol
Server Interface
Queuing, Scheduling, and Activation Control
File System Metadata Services
Tornado Coded Stripe Storage
Block Location Metadata Services
Abstract Storage Interface
Block or Object Access Protocol
Object Store
Object Location
Object Store
Object Location
Client Interface
Tornado Coding
File System Metadata
Block Storage

Figure 1. Tornado Coded Archival Storage software architecture

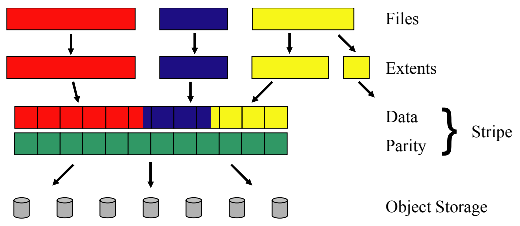


Figure 2. Tornado Coded Archival Storage data flow – an extent-based log-structured encoded-stripe file system

3.1. Tornado coded file system

The TCAS file system is an extent-based, log-structured, encoded-stripe file system (see Figure 2). Files are broken into extents as necessary and aggregated on fixed-size stripes. When a stripe is full, or a predefined write buffer timeout has expired, the entire stripe is encoded and the data and check nodes are distributed to storage devices. Any block-based or object-based storage target is an acceptable backing store.

The foundation of the Tornado Coded stripe storage is provided by a series of Tornado Code graphs with known failure profiles (see [10]). Each stripe consists of 48 data nodes and 48 check nodes. Even though this is close to the region where LDPC codes have high overhead, it is an appropriate lower bound because it allows several stripes to be read or written simultaneously while being small enough to facilitate explicit device management. The amount of data in each node (the block size) is a design choice – we chose to maintain a constant block size of 1 MB, so each stripe stores 48 MB of user data. This is close to the average size of files from our archival storage system trace data. However, the TCAS software supports variable block sizes at the stripe level, and the block size may be dynamically changed at runtime.

3.2. Block placement

Wide-scale distributed and peer-to-peer storage systems have typically used overlay networks or complex algorithms to place and locate data on a large number of devices. A MAID system, on the other hand, has finite capacity and is managed by a local system with complete control, so nodes must explicitly be placed and retrieved at runtime. By using predefined placement policies, the number of device activations to retrieve data can be minimized and failure sets more easily enumerated.

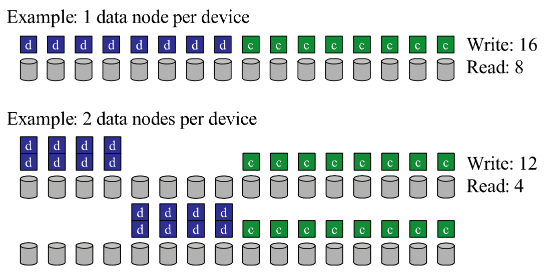

Example: 1 data node per device
Write: 16
Read: 8
Example: 2 data nodes per device
Write: 12
Read: 4

Figure 3. Example overloaded data node placement

To simplify the placement problem, we have subdivided the disks in the backing store into a number of *placement groups* that are the size of Tornado Coded stripes. For example, a 960-device MAID unit may be configured with 10 placement groups of 96 disks each. One placement group remains active to accept writes, while others may be activated or deactivated for reads. More complex layouts, such as spiraling stripes throughout the system, may be achieved using this technique without the scalability problems that would arise using metadata to associate every node to a device. This also provides flexibility for easily remapping failed disks, using disks with different capacities, and increasing the system's capacity at runtime.

3.3. Overloaded block placement

One way of attempting to improve performance is by increasing parallelism. In the case of TCAS on MAID, device parallelism is already present: every write accesses 96 devices and every read accesses 48 devices. One way to increase operational parallelism is by overloading data nodes. Placing multiple data nodes on a single device reduces the number of devices that must be accessed for each stripe and thus increases the number of stripe operations that may be performed in parallel. We examined this technique only on data nodes. Overloading the check nodes would be possible, but because check nodes are not normally retrieved, overloading them would decrease fault tolerance for little benefit.

At first glance, overloading data nodes is counterintuitive because fault tolerance is achieved by using large-scale LDPC codes and independent devices. Our results show that placing two or three data nodes on a device decreases the fault tolerance of Tornado Coded stripes by a small amount – from theoretical MTBF values in the billions of years to millions (see Table 1). (In this configuration, mirroring has a reciprocal reliability of only 208.8 years, so even the worst overlapped Tornado Codes have higher reliability than mirroring.) For several overload policies and Tornado Code graphs, the first failure does not decrease at all. Overloading becomes a matter of policy: a system designer may choose to overload data nodes to help performance, or omit it to obtain the highest fault tolerance possible using a particular Tornado Code graph.

Table 1 – MTBF (reciprocal of probability of failure) in millions of years for 96-node storage systems by data node placement policy with independent disk failures with an annual failure rate p=.01 and no repair

Overload Policy	Tornado Graph 1	Tornado Graph 2	Tornado Graph 3
1:1 – 96 None	743.49	1681.52	1707.36
2:1 – 72 Block	23.61	44.13	425.35
2:1 – 72 Cyclic	13.58	314.76	81.30
3:1 – 64 Block	6.89	6.59	20.35
3:1 – 64 Cyclic	0.91	7.49	0.48

4. Trace-based workload simulation

The target system architecture consists of a control system and MAID storage array connected with a high-throughput I/O interconnect (see Figure 4). In the simulated configuration, the MAID array contains 960 drives and the percentage of drives that are allowed online at any time is an independent variable. Drive spinup operations require 10 seconds and are normally distributed, and the maximum disk throughput is limited to 50 MB/s for a 1 MB operation. The Fibre Channel interconnect is limited to 200 MB/s. We believe these values conservatively represent expected performance of a MAID system constructed with currently available commodity disk drives and interconnects.

The simulated control system consists of a single multiprocessor computer with a variable number of CPUs and amount of system memory. In the target design, three processors are dedicated to encoding and decoding stripes. System memory is another of the limited resources required for TCAS operations. A stripe requires 96 MB of memory, so the number of stripes that may be allocated memory is controlled. The scheduler maintains two queues each for write and read operations. The first queue collects requests that must wait for resources, and a second queue contains stripes that have been allocated memory. This separates device activations and planning from data movement and coding operations.

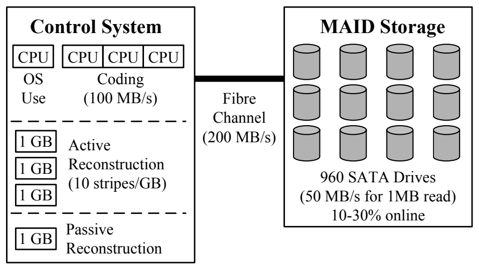


Figure 4. Organization and resource constraints for Tornado Coded MAID system emulator

The scheduler prevents the starvation of requests by ordering activations and prioritizing block read and write commands. Read and write operations are scheduled before activations and deactivations to prevent disk activation thrashing. The longer a stripe has been waiting, the more it influences device activations to retrieve necessary blocks. Also, after every block retrieval operation, the Tornado Code reconstruction algorithm is run on the stripe's metadata state to prevent nodes that can be reconstructed from being unnecessarily retrieved.

4.1. Evaluation traces and metrics

To generate traces for simulation, we started with sanitized log excerpts of file access patterns from a multi-petabyte tape-based archival storage system. Access data from two periods of time was processed to create simulation preload and analysis sets. The first two days of data were used to preload the file system, and the final 24 hours became the trace for simulation. The preloaded file systems contain about 100,000 files and 10 TB of data, and the traces simulate reading and writing about 2 TB of data in 25,000 files. The average read and write throughputs of 24 and 31 MB/s respectively are under the simulated capacity so queues do not grow indefinitely.

Because the system is driven by user requests, and users hate to wait, the amount of time required to deliver a file is our preferred performance metric. For all of our experiments, each configuration was implemented and the file system preloaded. Then, for each parameter of interest, at least three runs of the trace were performed. The average file operation times were calculated, and the results of the runs were averaged and plotted.

4.2. Simulation results

As expected, both increasing the number of data nodes per device and increasing the maximum number of devices allowed online reduce the amount of time required to retrieve a file (see Figure 5). The initial cases with low parallelism have the poorest performance, and as parallelism increases, the read response times approach the device activation time of 10 seconds.

Increasing the data node overlap and online device envelope generally reduces the mount count (see Figure 6), but one pathological case was found. When the no-overlap case transitions from 144 to 192 devices online, the mount count dramatically increases. In the 144 online case, with one data node per device, only one stripe can be read at a time. By policy, the active devices cannot be powered down until there are no stripes requiring those devices present in the reconstruction queue. In the 192 online case, two stripes can be retrieved simultaneously, and the additional flexibility is immediately leveraged to activate more drives. The mount count increases dramatically, and the wait time decreases.

Our results show that the time required to write a file to archival storage is consistent across all configurations for a particular trace, but varies between traces. The consistency across configurations is expected, as the system always writes one stripe at a time to ready devices as the log is flushed. Between traces, the difference appears to be an interaction of the bus throughput and the test case itself. For example, if one trace writes a few very large files in a short period of time, these large file write operations would become a large number of stripe write operations and greatly increase the average write response time across all configurations running that trace.

4.3. Discussion

The placement policies and device constraints clearly affect performance and mount activity. For read performance, it is possible to configure the system to a break-even point of read parallelism where additional increases in data node overloading or devices online do not provide additional gain. By simply allowing enough devices online it is possible to achieve an average file read time approximately equivalent to the device activation time. Placing more devices per data node or allowing more devices online does not improve performance, but does decrease the device mount count.

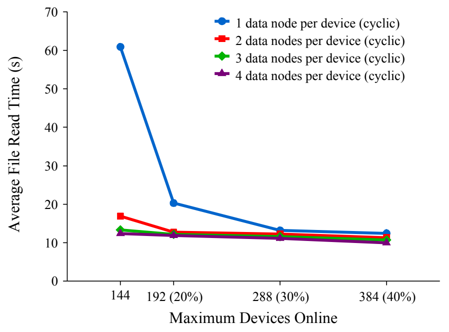


Figure 5. Simulated average file read time by maximum devices online and placement policy

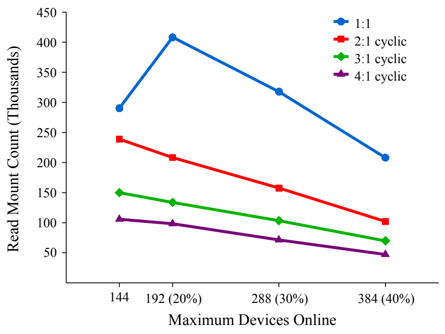


Figure 6. Simulated read mount count by maximum devices online and data node placement policy

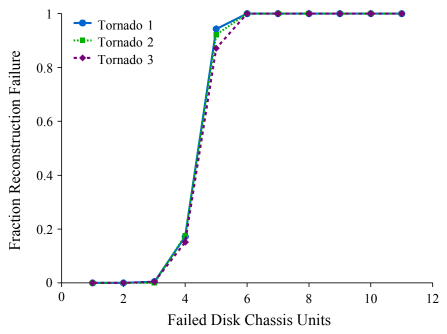


Figure 7. Fraction reconstruction failure by number of failed disk chassis units in the Maelstrom system

The TCAS MAID simulation provides a convenient mechanism for experimenting with policy changes in a large-scale storage system. Not only does the simulator provide file access statistics, but it calculates device activation operations and estimates total power usage. By combining the simulated performance with fault tolerance data, a system designer can choose parameters to create a system with the desired performance and fault tolerance.

5. Prototype implementation

Simulation is a logical preliminary step in the design and evaluation of any complex system, but eventually the short but important question arises: Does it work? The software developed for the TCAS MAID evaluation serves as both a discrete-event simulator and a functional prototype. In simulation mode, the software goes through the motions of manipulating blocks through an event-driven data flow. The same code also supports emulating a fully functional archival storage system, performing all the data manipulation required to encode, store, decode, and retrieve files in response to interactive user requests.

For preliminary testing, the simulation was extended to support uploading and downloading real files, and we tested archiving and retrieving entire local Linux installations. Fault tolerance was examined by deleting subdirectories containing the encoded data for entire devices. These tests confirmed that our TCAS software can store, retrieve, and reconstruct real data.

We chose to expose the TCAS system using the GridFTP interface because of its ubiquity in the scientific computing and data grid communities. We wrote a custom GridFTP data storage interface (DSI) module to serve as the TCAS front end. Our DSI is based on the Globus sample source code, but we thoroughly examined the SRB DSI [1] to become familiar with DSI development techniques. We loosely coupled the DSI and the back-end storage system using a SQL database and a storage cache (see Figure 9). The loose coupling regulates the flow of data to and from the slower archival backing

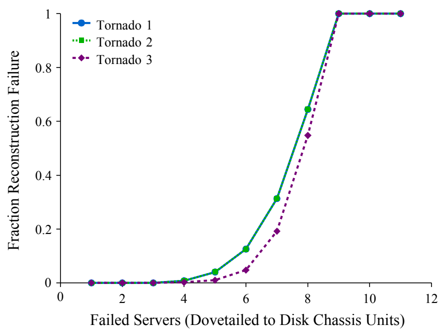


Figure 8. Fraction reconstruction failure by number of failed servers in the Maelstrom system

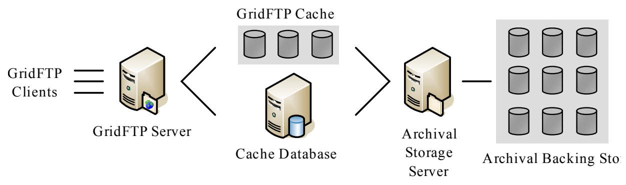


Figure 9. TCAS system and GridFTP DSI coupling

store. In the prototype implementation, all operations are atomic and sent to archival storage for fulfillment.

5.1. Storage cluster deployment

We deployed the TCAS system on NCAR's 100 TB Lustre storage cluster named Maelstrom. The cluster consists of 12 storage servers, each of which manages a single disk array configured with RAID5 LUNs. Because the Tornado Codes have 96 nodes, and the storage cluster only has 12 independent storage devices, a cyclic 96:12 node to device mapping was used to associate nodes with devices. Nodes are represented by appropriately named subdirectories that have been configured to stripe across only one object storage target (OST). We verified that the TCAS software on Maelstrom was able to upload and download files from other machines using GridFTP.

To examine the effect of deploying TCAS on this system we enumerated disk chassis and server failure case scenarios through simulation. The reliability of the system describes the potential of data loss given the failure of an increasing number of disk chassis units (see Figure 7). All three of the Tornado Code graphs we tested can survive the loss of two disk chassis units, and one can survive the loss of any three. If the servers and disk units are properly dovetailed in a high availability configuration, the TCAS storage remains available with the failure of any three storage servers (see Figure 8). In fact, if servers are chosen appropriately, half of the storage servers in Maelstrom can be taken offline for maintenance without causing loss of availability. The primary benefit of this implementation is that it takes advantage of Lustre to gracefully handle device failures and to provide access to the backing store.

6. Future work

There is a broad range of opportunities for continued work in the area of Tornado Coded archival storage. For fault tolerance, additional work may provide more accurate system reliability based on expected workloads. Our initial Tornado Code graph fault tolerance investigations assumed a device annual failure rate (AFR) of 1.0%. This AFR does not reflect the device power state changes performed by a MAID system, which is important because powering drives on and off several times per hour influences drive reliability. Our simulator is capable of producing device activation data based on archival storage access traces. To close the reliability loop, this simulated device activation data could be used to produce time-variant device AFR values, and then to calculate system reliability based on device reliability over time.

The prototype implements the data flow, but as with most research prototype log-structured file systems, many nontrivial details such as stripe cleaning and file version interfaces have not been completed. For power-aware MAID configurations, switching from hard constraints to soft constraints could allow the activation of more drives during high usage periods to reduce file delivery time. We believe that it is possible to identify an optimal power envelope given a hardware configuration and real-time system load.

7. Conclusion

Tornado Codes provide erasure coding with probabilistically successful data reconstruction. In this paper, we presented an architecture for a MAID storage system using Tornado Codes for fault tolerance in an archival storage system. Through simulation, we have examined several design considerations for such of a system. Our results show that a simple block-based data layout, possibly placing multiple data nodes per device, provides the best performance. The system can easily be deployed on traditional hardware and exposed using a standard interface. In the future, we plan to extend TCAS to construct a federated storage system within a data grid infrastructure. By combining Tornado Codes and MAID, the system will provide highly reliable data storage within a configurable power footprint.

Acknowledgements

We would like to thank Dirk Grunwald for his suggestion of using Tornado Codes with MAID storage systems and Manish Vachharajani for the idea of overloading data node placement. Adam Boggs, Jason Cope, Sean McCreary, Michael Oberg, and Theron Voran provided valuable feedback. Support and computer time were 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 program.

References

[1] Bresnahan, John. SRB GridFTP DSI (Computer Software Source Code), 11 October 2006. http://www.mcs.anl.gov/~bresnaha/globus_srb_dsi-0.13.tar.gz

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

[3] Colarelli, D., and D. Grunwald, "Massive Arrays of Idle Disks for Storage Archives", Proceedings of the 2000 ACM/IEEE Conference on Supercomputing (SC'02), 1-11, November 2002.

[4] Delco, M., H. Weatherspoon, and S. Zhuang, "Typhoon: An Archival System for Tolerating High Degrees of File Server Failure", University of California, Berkeley project report, 13 December 1999. http://www.cs.berkeley.edu/~hweather/Typhoon/

[5] Luby, M. G., M. Mitzenmacher, M.A. Shokrollahi, and D.A. Spielman, "Efficient Erasure Correcting Codes", IEEE Transactions on Information Theory, 47(2), 569-584, February 2001.

[6] Moore, R.W., J.F. JaJa, and R. Chadduck, "Mitigating Risk of Data Loss in Preservation Environments," Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005), 2005.

[7] Plank, J.S., A.L. Buchsbaum, R.L. Collins, and M.G. Thomason, "Small Parity-Check Erasure Codes - Exploration and Observations", Proceeding of DSN-05: International Conference on Dependable Systems and Networks, Yokohama, Japan, June 2005.

[8] Wan, M., A. Rajasekar, R. Moore, and P. Andrews, "A Simple Mass Storage System for the SRB Data Grid", Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems & Technologies (MSST 2003), San Diego, CA, April 2003.

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

[10] Woitaszek, M. and H. M. Tufo, "Fault Tolerance of Tornado Codes for Archival Storage", Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC 15), Paris, France, June 2006.

[11] Xia, H. and A.A. Chien, "RobuSTore: Robust Performance for Distributed Storage Systems", Proceedings of the 14th NASA Goddard - 23rd IEEE Conference on Mass Storage Systems and Technologies (MSST 2006), May 2006.