

# Assessing the Performance of Erasure Codes in the Wide-Area

Rebecca L. Collins

James S. Plank

Department of Computer Science  
University of Tennessee  
Knoxville, TN 37996  
rcollins@cs.utk.edu, plank@cs.utk.edu

Appearing in:

*DSN-2005: The International Conference on Dependable Systems and Networks*  
IEEE, Yokohama, Japan, June 2005.

<http://www.cs.utk.edu/~plank/plank/papers/DSN-2005-CP.html>

# Assessing the Performance of Erasure Codes in the Wide-Area

Rebecca L. Collins

James S. Plank

Department of Computer Science

University of Tennessee

Knoxville, TN 37996

[rcollins,plank]@cs.utk.edu

## Abstract

*The problem of efficiently retrieving a file that has been broken into blocks and distributed across the wide-area pervades applications that utilize Grid, peer-to-peer, and distributed file systems. While the use of erasure codes to improve the fault-tolerance and performance of wide-area file systems has been explored, there has been little work that assesses the performance and quantifies the impact of modifying various parameters. This paper performs such an assessment. We modify our previously defined framework for studying replication in the wide-area [6] to include both Reed-Solomon and Low-Density Parity-Check (LDPC) erasure codes. We then use this framework to compare Reed-Solomon and LDPC erasure codes in three wide-area, distributed settings. We conclude that although LDPC codes have an advantage over Reed-Solomon codes in terms of decoding cost, this advantage does not always translate to the best overall performance in wide-area storage situations.*

## 1. Introduction

The coordination of widely distributed file servers is a complex problem that challenges wide-area, peer-to-peer and Grid file systems. Systems like OceanStore [16], LoCI [3], and BitTorrent [5] aggregate wide-area collections of storage servers to store files on the wide-area. The files are broken into fixed-size blocks, which are distributed among the disparate servers. To address issues of locality and fault-tolerance, *erasure coding* is employed, albeit normally in the primitive form of block replication. When a client needs to download a file from this collection of servers to a single location, he or she is faced with a variety of decisions to make about how to perform the download. This has been termed the “Plank-Beck” problem by Allen and Wolski, who deemed it one of two fundamental problems of data movement in Grid Computing systems [1].

In previous work, we defined a framework for evaluating

algorithms for downloading replicated wide-area files [6]. In this paper, we extend this framework to include erasure coding, and perform an evaluation of algorithms for downloading erasure-encoded files in three wide-area settings. This work is significant as it is the first piece of work to evaluate the relative performance of Reed-Solomon codes and LDPC codes in a realistic, wide-area environment with aggressive, multithreaded downloading algorithms. The two types of erasure codes present different downloading and decoding trade-offs, and in most cases, Reed-Solomon codes perform well when sets are small and the client’s connection to the network is limited – situations likely to arise in wide-area storage systems.

## 2. The “Plank-Beck” Problem

We are given a collection of storage servers on the wide-area, and a large file that we desire to store on these servers. The file is partitioned into blocks (termed “data blocks”) of a fixed size, and an erasure coding scheme is used to calculate some number of additional “coding” blocks. The collection of data and coding blocks are then distributed among the storage servers. At some point, a client at a given network location desires to download the file, either in its entirety, or in a streaming fashion. Simply stated, the “Plank-Beck” problem is: How does the client download the file with the best performance.

The simplest erasure coding scheme is replication – each block of the file is stored at more than one server. In this case, algorithms for solving the “Plank-Beck” problem may be parameterized in four dimensions [6]:

1.  $T$ , the number of simultaneous downloads.
2.  $R$ , the degree of *work replication*; that is, what is the maximum number of simultaneous downloads of a block?
3.  $P$ , the *failover strategy*, or how many blocks must be retrieved after a given block’s first download begins before that block requires replication?

4. The server selection strategy. We focus on the two best performing server selection algorithms with replication from [6], called **fastest<sub>0</sub>** and **fastest<sub>1</sub>**. **Fastest<sub>0</sub>** always selects the fastest server. **Fastest<sub>1</sub>** adds a “load number” for the number of threads currently downloading from each server, and selects the server that minimizes this combination of predicted download time and load number.

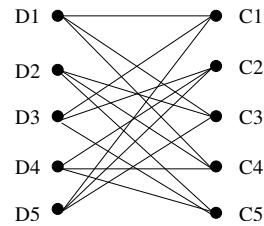
The optimal choice of parameters depends on many factors, including the distribution of the file and the ability for the client to utilize monitoring and forecasting [19]; however, a few general conclusions were drawn. First, the number of simultaneous downloads should be large enough to saturate the network paths to the client, without violating TCP-friendliness in shared networks. Second, work replication should be present, but not over-aggressive; ideally, it combines with a failover strategy that performs work replication when the progress of the download falls below a threshold. Finally, if monitoring software is present, fast servers (with respect to the client) should be selected, although ideal algorithms consider both the forecasted speed from server and the load induced on the server by the download itself. When we add more complex erasure coding, some additional parameters manifest themselves. We define them after explaining erasure codes in more detail.

### 3. Erasure Coding

Erasure codes have arisen as a viable alternative to replication for both caching and fault-tolerance in wide-area file systems [4, 15, 17, 20]. Formally defined, with erasure coding,  $n$  data blocks are used to construct  $m$  coding (or “check”) blocks, where data and check blocks have the same size. The encoding *rate* is  $\frac{n}{n+m}$ . Subsets of the data and coding blocks may be used to reconstruct the original set of data blocks. Ideally, these subsets are made up of any  $n$  data or check blocks, although this is not always the case. Replication is a very simple form of erasure encoding (with  $n = 1$  and  $m > 1$ ). In comparison to replication, more complex erasure coding techniques (with  $n > 1$  and  $m > 1$ ) reduce the burden of physical storage required to maintain high levels of fault tolerance. However, such erasure coding can introduce computationally intensive encoding and decoding operations. We explore the two most popular types of erasure coding in this paper: Reed-Solomon coding, and Low-Density Parity-Check (LDPC) coding. They are summarized below.

#### 3.1. Reed-Solomon Coding

Reed-Solomon codes have been used for fault-tolerance in a variety of settings [7–9, 11, 20]. Their basic operation



**Figure 1. An example LDPC code where  $n = 5$  and  $m = 5$**

is as follows. The  $n$  data blocks are partitioned into *words* of a fixed size, and a collection of  $n$  words forms a vector. A *distribution matrix* is employed so that the check blocks are calculated from the vector with  $m$  dot products. Galois Field arithmetic is used so all elements have multiplicative inverses. Then the act of decoding is straightforward. Given any  $n$  of the data and check blocks, a decoding matrix may be derived from the distribution matrix, and the remaining data blocks may be calculated again with dot products.

A tutorial on Reed-Solomon coding is available in [12, 14]. Reed-Solomon coding is expensive for several reasons. First, the calculation of check block requires an  $n$ -way dot product. Thus, creating  $m$  check blocks of size  $B$  has a complexity of  $O(Bmn)$ , which grows rather quickly with  $n$  and  $m$ . Second, decoding involves an  $O(n^3)$  matrix inversion. Third, decoding involves another  $n$ -way dot product for each data block that needs to be decoded. Fourth, Galois-Field multiplication is more expensive than integer multiplication. For that reason, Reed-Solomon coding is usually deemed appropriate only for limited values of  $n$  and  $m$ . See [4, 15] for some limited evaluations of Reed-Solomon coding as  $n$  and  $m$  grow.

#### 3.2. LDPC Coding

LDPC coding is an interesting alternative to Reed-Solomon coding. LDPC codes are based on bipartite graphs, where the data blocks are represented by the left-hand nodes, and the check blocks are represented by the right-hand nodes. An example for  $n = 5$  and  $m = 5$  is given in Figure 1. Each check block is calculated to be the bitwise exclusive-or of all data blocks incident to it. Thus, for example, block  $C1$  in Figure 1 is equal to  $D1 \oplus D3 \oplus D5$ . A check block can only be used to reconstruct data blocks to which it is adjacent in the graph. For example, check block  $C1$  can only be used to reconstruct data block  $D1$ ,  $D3$ , or  $D5$ . If we assume that the blocks are downloaded randomly, sometimes more than  $n$  blocks are required to reconstruct all of the data blocks. For example, suppose blocks  $D3, D4, D5, C2$ , and  $C5$  are retrieved. There is no way to reconstruct block  $D1$  with these blocks, and a sixth

**Table 1. Experiment space**

Parameter	Range of Parameters
Simultaneous Downloads	$T \in [20, 30]$
Work Replication and Failover Strategy	$\{R, P\} \in [\{1, 1\}, \{2, (30/n)\}]$
Server Selection	<b>fastest<sub>0</sub>, fastest<sub>1</sub></b>
Block Selection	<b>db-first, no-pref</b>
Coding	$n \in [5, 10, 20, 50, 100]$ , $m = n$

block must be retrieved. Furthermore, if the next block randomly chosen is  $D2$  (even though it can be decoded with blocks  $D3, D4$ , and  $C5$ ), then it is still impossible to reconstruct block  $D1$  and a seventh block is required. A graph  $G$ 's overhead,  $o(G)$ , is the average number of blocks required to decode the entire set, and its *overhead factor*  $f(G)$  is equal to  $o(G)/n$ .

Unlike Reed-Solomon codes, LDPC codes have overhead factors greater than one. However, LDPC codes have a rich theoretical history which has shown that infinitely sized codes of given rates have overhead factors that approach one [10, 18]. For small codes ( $n \leq 100$ ), optimal codes are in general not known, but a recent exploration has generated codes whose overhead factors are roughly 1.15 for a rate of  $\frac{1}{2}$  [15]. We use selected codes from this exploration for evaluation in this paper.

## 4. Experimental Setup

We performed a set of experiments to compare Reed-Solomon and LDPC codes. The experiments were “live” experiments on a real wide-area network. The storage servers were Internet Backplane Protocol (IBP) servers [13], which serve time-limited, shared writable storage in the network. The actual servers that we used were part of the Logistical Backbone [2], a collection of over 300 IBP servers in various locations worldwide.

In our experiments, we stored a 100 MB file, decomposed into 1 MB blocks plus coding blocks, and distributed them on the network. Then we tested the performance of downloading this file to a client at the University of Tennessee. The client machine ran Linux RedHat version 9, and had an Intel (R) Celeron (R) 2.2 GHz processor. Since the downloads took place over the commodity Internet, the tests were executed in a random order so that trends due to local or unusual network activity were minimized, and each data point presented is the average of ten runs.

Table 1 shows the parameter space explored in the experiments. A block selection criteria based on the type of block is introduced, and will be detailed in section 5.1. The actual codes used can be found in the

online appendix ([www.cs.utk.edu/~rcollins/papers/CS-04-536\\_Appendix.html](http://www.cs.utk.edu/~rcollins/papers/CS-04-536_Appendix.html)). They were derived as a part of the exploration in [15], and therefore are not provably optimal.

### 4.1. Network Files

We employ three network files in the experiments. These differ by the way in which they are distributed on the wide-area network. The first is called the **Hodgepodge distribution**. In this file, fifty regionally distinct servers are chosen and the data and check blocks are striped across these fifty servers. None of the servers is in Tennessee, the location of the client. The next two network files are the **Regional distribution** and the **Slow Regional distribution**. In these files, the data and check blocks are striped across servers in four regions. The regions chosen for the Regional distribution are Alabama, California, Texas, and Wisconsin; and the regions chosen for the Slow Regional distribution are Southeastern Canada, Western Europe, Singapore, and Korea.

## 5. Reed-Solomon vs. LDPC

When comparing the two types of coding, the properties of key importance are the encoding and decoding times, and the average number of blocks that are necessary to reconstruct a set. LDPC codes have a great advantage in terms of encoding and decoding time over Reed-Solomon codes; in addition, Reed-Solomon decoding requires  $n$  blocks from a set before decoding can begin, while LDPC decoding can take place on-the-fly. However, for small  $n$ , and in systems where the network connection is slow, Reed-Solomon codes can sometimes outperform LDPC codes despite the increased decoding penalty [15].

### 5.1. Subtleties of LDPC Implementation

The implementation of LDPC coding involves some subtleties that are not present in that of Reed-Solomon coding. For example, the fact that LDPC coding sometimes requires more than  $n$  blocks affects not only *how many* blocks must be retrieved, but also limits *which* blocks the client application can choose. A wide range of block scheduling strategies may be applied to an LDPC coding set. At one extreme, blocks are downloaded randomly, until enough blocks have been retrieved to decode the set. It is likely that some of the coding blocks will become useless by the time they are retrieved, and that some data blocks may be decoded before they are retrieved. At the other extreme, a download may be simulated in order to determine an “optimal” set of blocks that can be used to decode the set. The difficulty here is that it must be determined up front which blocks are coming from the fastest servers. Any optimal schedule would

**Table 2. Approximate overheads of LDPC codes used in the experiments**

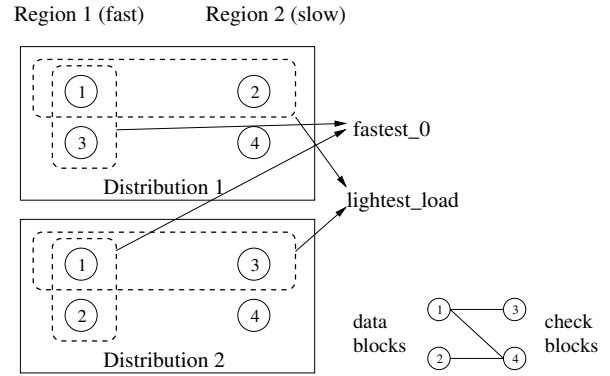
$n,m$	overhead factor (avg # blocks)	improved overhead factor
5,5	1.1071 (5.5355)	1.0690 (5.3450)
10,10	1.1440 (11.440)	1.0620 (10.620)
20,20	1.1971 (23.942)	1.1233 (22.466)
50,50	1.2795 (63.975)	1.1948 (59.740)
100,100	1.1477 (114.77)	1.0733 (107.33)

have to approximate the load of servers not only throughout the download of a given set, but between the downloads of different sets in the file, since they often overlap. The following experiments use a compromise between the two extreme scheduling options: when selecting a block to download, the downloading algorithm will skip over check blocks that can no longer contribute to decoding and data blocks that are already decoded. The algorithm also allows one of two block preferences to be specified:

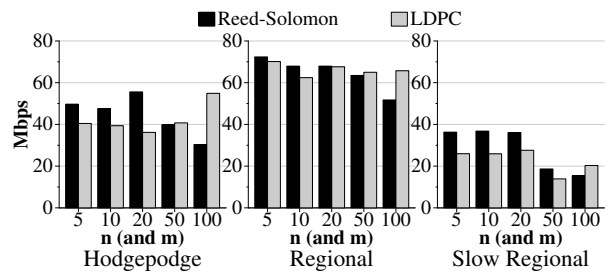
- Data blocks first (**db-first**): data blocks are always preferred over check blocks.
- No preference (**no-pref**): the type of blocks is ignored, and blocks are chosen solely based on the speed and load of the servers on which they reside.

Since decoding time and download time are the most important factors in these experiments, in comparing Reed-Solomon and LDPC codes, we would like to answer the two following questions: *How long does decoding take?* and *How many blocks do we need to retrieve?* To address the second question, recall that Reed-Solomon codes have optimal encodings; that is, sets may be decoded after any  $n$  blocks are retrieved. LDPC codes do not have optimal encodings. Table 2 lists the average overheads of the LDPC codes used in our experiments. Since we also use limited scheduling, the actual overheads in the experiments are slightly improved, and are also listed in Table 2. The overhead factor for the  $n = 5, m = 5$  code was calculated directly, and the rest of the values were produced by Monte Carlo simulations over 10,000,000 random downloads. These overheads represent the number of blocks required with the **no-pref** block selection strategy.

Another subtlety in the implementation of LDPC coding is that the distribution of the file can impact the performance of different server scheduling algorithms. Consider the example depicted in Figure 2, where  $n = 2$ , and  $m = 2$ , and there are two server regions. Figure 2 shows two possible distributions of the blocks, and which blocks would be chosen from each distribution by the **fastest<sub>0</sub>** and **lightest-load** server scheduling algorithms [6], where the **lightest-load**



**Figure 2. The performance of server scheduling algorithms can vary depending on the distribution of a file with LDPC coding.**



**Figure 3. Reed-Solomon vs. LDPC coding (performance including both download and decoding times)**

algorithm always chooses the server with the lightest load. Depending on the distribution, one of the algorithms results in two blocks that can be used to reconstruct the entire set, and the other does not. The unfortunate consequence of this characteristic is that the performance can vary greatly between different server scheduling algorithms not because of the algorithms themselves, but because of the file’s distribution. The following experiments use the **fastest<sub>0</sub>**, and **fastest<sub>1</sub>** server scheduling algorithms, and the distributions described in section 4.1, which do not address the distribution subtleties of LDPC coding – it is a subject of further work to address the impact of these subtleties on download performance.

## 5.2. Broad Trends

The best performing instances of Reed-Solomon and LDPC coding are shown in Figure 3. LDPC coding performs no better than, and in some cases much worse than Reed-Solomon coding when  $n$  is less than 50; however, when  $n$  is greater than 50, LDPC coding outperforms Reed-

Solomon coding. As  $n$  increases past 20, the performance of Reed-Solomon coding steadily declines, while the performance of LDPC coding tends to improve, as in the Hodgepodge distribution, or level off, as in the Regional distribution. Concerning the best overall performance, the best data point over all set sizes in the Hodgepodge distribution occurs at  $n = m = 100$  for LDPC coding, while in both the Regional and the Slow Regional distributions, Reed-Solomon achieves the best data point over all at  $n = m = 5$  and  $n = m = 10$ , respectively.

When judging the merits of either type of coding scheme in this particular application, it is important to remember that any size set can scale to arbitrarily large files without incurring additional decoding overhead per set. In general, given an application and a choice between Reed-Solomon or LDPC coding it is probably best to choose the scheme and set  $n, m$  that:

- Is able to scale in the future along with the application.
- Does not exceed physical storage limitations.
- Meets desired levels of fault tolerance; note that Reed-Solomon coding has stronger guarantees than LDPC coding in this respect.
- Satisfies each of the three previous criteria with the best performance where performance consists of both download time and decoding time.

### 5.3. Block Preferences

In these experiments, blocks can be downloaded selectively based on their type. Data blocks are generally preferable to check blocks since fewer data blocks require less decoding, but when check blocks are much closer than data blocks, sometimes the advantage of getting close blocks is worth the decoding penalty. Table 3 shows which block preference algorithm is used in the best performing instances of the codes over the three distributions. A trade-off between download time and decoding time is apparent: the **no-pref** algorithm is favored in the Slow Regional distribution and in the Hodgepodge distribution, since the penalty of retrieving slow blocks is greater than the computational cost of decoding. In the Regional distribution, the **db-first** algorithm is favored — since the majority of blocks are closer to the client, the penalty for selecting blocks that require no decoding is small. In addition, when  $n \geq 50$  (i.e., the decoding time is greater), the **db-first** algorithm is slightly favored. To illustrate this trade-off further, Figures 4 and 5 show the total time spent downloading and decoding in the best performing instances of each distribution. Note that although one might think that the computational overhead would be similar for equal values of  $n$  and  $m$ , in the Regional distribution, the computational overhead is lower than the rest because the fastest instances prefer data

Table 3. Block preferences

Distribution $n, m$	LDPC preference	RS preference
Slow Regional 5,5	<b>no-pref</b>	<b>no-pref</b>
Slow Regional 10,10	<b>no-pref</b>	<b>no-pref</b>
Slow Regional 20,20	<b>no-pref</b>	<b>no-pref</b>
Slow Regional 50,50	<b>db-first</b>	<b>no-pref</b>
Slow Regional 100,100	<b>no-pref</b>	<b>no-pref</b>
Hodgepodge 5,5	<b>no-pref</b>	<b>no-pref</b>
Hodgepodge 10,10	<b>no-pref</b>	<b>no-pref</b>
Hodgepodge 20,20	<b>no-pref</b>	<b>no-pref</b>
Hodgepodge 50,50	<b>db-first</b>	<b>db-first</b>
Hodgepodge 100,100	<b>no-pref</b>	<b>db-first</b>
Regional 5,5	<b>db-first</b>	<b>db-first</b>
Regional 10,10	<b>db-first</b>	<b>db-first</b>
Regional 20,20	<b>db-first</b>	<b>no-pref</b>
Regional 50,50	<b>db-first</b>	<b>db-first</b>
Regional 100,100	<b>db-first</b>	<b>db-first</b>

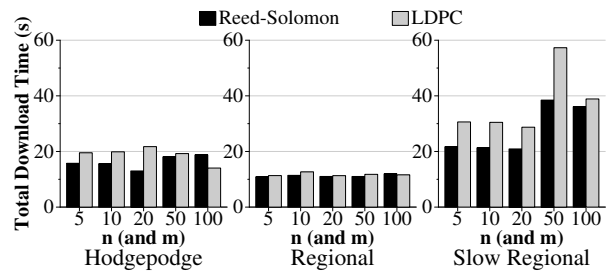
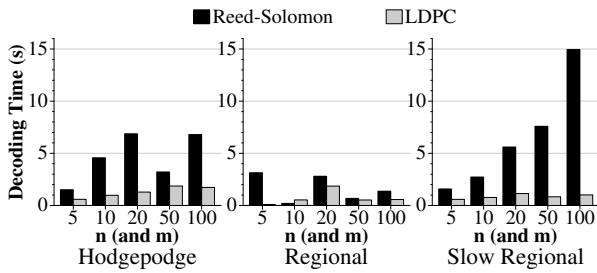


Figure 4. Total download time (beginning of first block download to ending of last) in best performances

blocks to coding blocks. In general, these figures clearly show that Reed-Solomon codes require more time to decode than LDPC codes. However, since the decoding of sets can overlap with the downloading of subsequent sets, the cost of decoding will negatively impact performance only when the decoding time surpasses the downloading time, and when the last sets of the file retrieved are being decoded.

## 6. Conclusions

When downloading algorithms are applied to a wide-area file system based on erasure codes, additional considerations must be taken into account. Performance reflects a balance between the time it takes to download the necessary number of blocks and the time it takes to decode the blocks that have been retrieved. In systems where the client has very fast connections to the servers, the decoding time is more expensive relative to the download time, and downloading schemes that preferentially choose data blocks will



**Figure 5. Total decoding time in best performances**

perform the best. On the other hand, in systems where the client has slow connections to the servers, the decoding time becomes relatively inexpensive and downloading schemes that choose the closest blocks regardless of block type will have the best performance.

This work compares Reed-Solomon erasure codes to LDPC erasure codes. The main limitation of Reed-Solomon codes is that decoding becomes prohibitively expensive in larger sets, while the limitation of LDPC codes is that their encoding is not optimal. LDPC codes outperform Reed-Solomon codes in large sets because LDPC decoding is very inexpensive, but in smaller sets which are more likely to be used in storage systems, Reed-Solomon codes outperform LDPC codes. In our experiments, Reed-Solomon codes performed better than or nearly equal to LDPC codes in data sets up to size 50 with a coding rate of  $\frac{1}{2}$ .

## 7. Acknowledgments

This material is based upon work supported by the National Science Foundation under grants CNS-0437508, ACI-0204007, ANI-0222945, and EIA-9972889. The authors thank Micah Beck and Scott Atchley for helpful discussions, and Larry Peterson for PlanetLab access.

## References

- [1] M. S. Allen and R. Wolski. The Livny and Plank-Beck Problems: Studies in data movement on the computational grid. In *Supercomputing 2003*, Phoenix, November 2003.
- [2] A. Bassi, M. Beck, T. Moore, and J. S. Plank. The logistical backbone: Scalable infrastructure for global data grids. In *Asian Computing Science Conference 2002*, Hanoi, Vietnam, December 2002.
- [3] M. Beck, T. Moore, and J. S. Plank. An end-to-end approach to globally scalable network storage. In *ACM SIGCOMM '02*, Pittsburgh, August 2002.
- [4] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data.

- In *ACM SIGCOMM '98*, pages 56–67, Vancouver, August 1998.
- [5] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003.
- [6] R. L. Collins and J. S. Plank. Downloading replicated, wide-area files – a framework and empirical evaluation. In *3rd IEEE International Symposium on Network Computing and Applications (NCA-2004)*, Cambridge, MA, August 2004.
- [7] P. J. Havinga. Energy efficiency of error correction on wireless systems. In *IEEE Wireless Communications and Networking Conference*, September 1999.
- [8] J. Kubiawicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.
- [9] W. Litwin and T. Schwarz. LH\* RS : A high-availability scalable distributed data structure using Reed Solomon codes. In *SIGMOD Conference*, pages 237–248, 2000.
- [10] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann. Practical loss-resilient codes. In *29th Annual ACM Symposium on Theory of Computing*, pages 150–159, El Paso, TX, 1997. ACM.
- [11] J. S. Plank. Improving the performance of coordinated checkpoints on networks of workstations using RAID techniques. In *15th Symposium on Reliable Distributed Systems*, pages 76–85, October 1996.
- [12] J. S. Plank. A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems. *Software – Practice & Experience*, 27(9):995–1012, September 1997.
- [13] J. S. Plank, A. Bassi, M. Beck, T. Moore, D. M. Swamy, and R. Wolski. Managing data storage in the network. *IEEE Internet Computing*, 5(5):50–58, September/October 2001.
- [14] J. S. Plank and Y. Ding. Note: Correction to the 1997 tutorial on Reed-Solomon coding. *Software – Practice & Experience*, 35(2):189–194, February 2005.
- [15] J. S. Plank and M. G. Thomason. A practical analysis of low-density parity-check erasure codes for wide-area storage applications. In *DSN-2004: The International Conference on Dependable Systems and Networks*. IEEE, June 2004.
- [16] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiawicz. Maintenance-free global data storage. *IEEE Internet Computing*, 5(5):40–49, 2001.
- [17] H. Weatherspoon and J. Kubiawicz. Erasure coding vs. replication: A quantitative comparison. In *First International Workshop on Peer-to-Peer Systems (IPTPS)*, March 2002.
- [18] S. B. Wicker and S. Kim. *Fundamentals of Codes, Graphs, and Iterative Decoding*. Kluwer Academic Publishers, Norwell, MA, 2003.
- [19] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5):757–768, October 1999.
- [20] Z. Zhang and Q. Lian. Reperasure: Replication protocol using erasure-code in peer-to-peer storage network. In *21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, pages 330–339, October 2002.