On the Speedup of Recovery in Large-Scale Erasure-Coded Storage Systems (Supplementary File)

Yunfeng Zhu, Patrick P. C. Lee, Yinlong Xu, Yuchong Hu, and Liping Xiang

✦

1 Additional Related Work

Our work focuses on the recovery solutions for XOR-based erasure codes. We point out that *regenerating codes* [5] have recently been proposed to minimize the recovery bandwidth in distributed storage systems. The idea is that surviving storage nodes compute and transmit linear combinations of their stored data during failure recovery. On the other hand, in XOR-based erasure codes, we do not require storage nodes be equipped with computational capabilities.

Authors of [6], [9], [14], [19] propose a new class of local recovery erasure codes that reduce recovery I/Os in distributed storage systems. The idea is to add additional parities (i.e., additional redundancy) to the existing stored data, so that recovery can be done by connecting to fewer than k surviving nodes. Instead of constructing new codes, our recovery solution builds on existing constructions of XOR-based erasure codes and preserve their data/parity layouts.

Recent work [25] discusses the failure recovery for XOR-based erasure codes on heterogeneous storage devices and proposes a cost-based heterogeneous recovery scheme for two RAID-6 (double-fault tolerant) codes RDP and EVENODD. The idea of the recovery scheme is to eliminate the search for the recovery solutions that are known to make no improvements to the resulting recovery performance, thereby improving the efficiency of the solution search process. Note that the search space of the recovery scheme in [25] remains exponential with respect to the number of nodes in the system, and it is still an open issue of how to extend the results of [25] for general XOR-based erasure codes.

- *Y. Zhu, Y. Xu, and L. Xiang are with School of Computer Science and Technology, University of Science & Technology of China (emails: {zyfl,xlping}mail.ustc.edu.cn, ylxu@ustc.edu.cn)*
- *P. P. C. Lee is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong (email: pclee@cse.cuhk.edu.hk)*
- *Y. Hu is with the The Institute of Network Coding, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong (email: yuchonghu@gmail.com).*

2 Motivated Scenarios

In the following, we describe several scenarios where enumeration recovery becomes *infeasible* to deploy, and hence motivate the need of replace recovery in such scenarios.

Limited hardware resources. Since the number of recovery equations increases exponentially with m and ω, enumeration recovery may consume substantial resources for enumerating all recovery equations when m and ω are large, which correspond to the codes with a large strip size and a higher level of fault-tolerance, respectively. For example, for the STAR code [10], we have $m = 3$ and $\omega = p-1$. If $p = 13$, then at most $2^{36}-1$ recovery equations need to be considered. Therefore, the search space can reach a maximum of $\binom{2^{36}-1}{12}$ combinations of recovery equations, and is too huge to enumerate. Our simulations (see the main file) show that with $m\omega \geq 20$, our enumeration recovery implementation deployed on commodity hardware cannot be finished within 13 days. Although one can use parallelization to speed up the enumeration process, the computational complexity remains exponential. Therefore, enumeration recovery becomes computationally infeasible when the available hardware resources are limited.

Remote recovery scenario. Nowadays, data recovery of a failed storage system can be outsourced to third-party companies (e.g., DataRecovery [3] and DataTech Labs [4]). However, the recovery service providers usually do not know *in advance* the coding scheme and parameters of the failed storage system. It is also difficult, while not impossible, to determine in advance the optimal solutions for all possible coding schemes and parameters. Thus, finding the optimal recovery solution for a failed storage system can be viewed as an on-demand decision task.

Heterogeneous recovery scenario. Enumeration recovery aims to minimize the number of read symbols for recovery. We can see that the optimal recovery solution is *deterministic*, meaning that for a given failed node, a coding scheme, and the corresponding parameters, the set of symbols being read from surviving nodes is fixed.

Thus, if m and ω are small, one can first determine the optimal solutions *offline* for each possible failed node. However, practical storage systems are typically composed of storage devices with heterogeneous capabilities (e.g., processing power, connectivity bandwidth) [12], [13]. In order to recover the failed node effectively, it is intuitive to retrieve fewer (more) data symbols from the surviving nodes with lower (higher) capabilities. Node capabilities may vary, depending on the current usage load of the storage system. Thus, it is important to find the optimal recovery solution *online* based on the current node capabilities. However, even it is possible to modify the enumeration recovery approach to account for the heterogeneous setting, its exponential complexity makes the online approach infeasible to capture the current node capabilities in a timely manner.

3 Simulations: Recovery Performance of HoRR

We evaluate the recovery performance of HoRR in a homogeneous setting. We aim to show that the number of read symbols returned by replace recovery is very close to that of enumeration recovery.

We consider different coding schemes under different fault tolerance levels. In the interest of space, we do not evaluate all coding schemes in the literature. We refer readers to [11] for the recovery performance of more coding schemes based on enumeration recovery. On the other hand, our goal is not to compare the recovery performance of different coding schemes; instead, we aim to compare different recovery schemes (i.e., conventional recovery, enumeration recovery, and replace recovery) for a specific coding scheme.

RAID-6 codes. We refer readers to the optimal recovery solutions of some RAID-6 codes including: RDP [23], EVENODD [21], and X-code [24]. Note that X-code is a vertical code and has no explicit parity nodes, but instead its parity symbols are placed in rows. To apply replace recovery, we create two imaginary parity nodes that correspond to the row parties (computed by the XOR-sums along diagonals of slope -1 and slope 1). We then apply replace recovery to the first $p-2$ lost data symbols in the failed strip, while the last two parity symbols of the failed strip are recovered by the corresponding data symbols used for encoding.

In general, for the RAID-6 codes (e.g., HDP [22]) where parity symbols are distributed among all nodes, we can also create imaginary parity nodes to apply our replace recovery. For example, for HDP [22], we can create two imaginary nodes that store the horizontal parity symbols and the anti-diagonal parity symbols, respectively.

Here, we focus on a class of *minimum density RAID-6 codes* [17], whose bit encoding matrices have minimum number of ones (i.e., non-zero entries) [17]. They can provide an optimal combination of different performance properties. Three constructions of minimum density RAID-6 codes are known: Blaum-Roth [1], Liberation [16] and Liber8tion [15]. Here, we show that our replace algorithm also provides optimal recovery performance for minimum density RAID-6 codes. We focus on Liber8tion, which has a strip size 8 and hence can be easily fit into blocks of practical file system. We implement Liber8tion, and evaluate via simulations the number of read symbols of both enumeration and replace approaches when a data node fails. Figure 1 shows the percentage of the number of read symbols compared to conventional recovery for different values of k (i.e., the number of data nodes). We observe that our replace algorithm achieves the same minimum number of read symbols as in enumeration recovery, and saves the number of read symbols by 26.56-29.17% compared to conventional recovery.

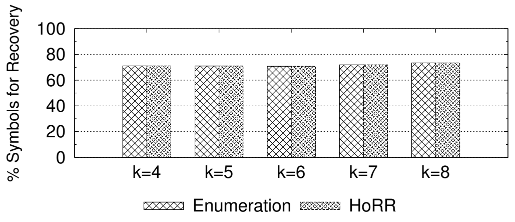


Fig. 1. Percentage of symbols per stripe needed for single-node failure recovery in Liber8tion compared to conventional recovery.

STAR codes. We now consider STAR [10]. For any prime p, STAR is composed of $p+3$ nodes, where the first p columns are data nodes and the remaining three columns are parity nodes. We first present a theorem that specifies the lower bound of the optimal recovery solution for STAR.

Theorem 3.1: The minimum number of read symbols for a single-node failure recovery for STAR is lower bounded by $(\frac{2}{3}p^2-p)$ (symbols per stripe).

Proof: Recall that a parity set is a set that contains a parity symbol and the data symbols encoded into that parity symbol. To recover a failed data node in STAR, we can select x parity sets of slope 0, y parity sets of slope -1, and z parity sets of slope 1. We can select $p-1$ parity sets from any combinations of x, y, and z, such that $x+y+z=p-1$. Thus, the number of symbols in the selected parity sets is $xp+(y+z)(p-1)$, in which the number of overlapping symbols is at most $x(y+z)+yz$. Thus, the number of read symbols for recovery per stripe (denoted by R) will be:

$$\begin{aligned}
R &\geq xp+(y+z)(p-1)-x(y+z)-yz\\
&= (p-1)p+(y+z)^2-p(y+z)-yz\\
&\geq (p-1)p+\frac{3}{4}(y+z)^2-p(y+z)\\
&= \frac{3}{4}[(y+z)^2-\frac{4}{3}p(y+z)+\frac{4}{9}p^2]+(p-1)p-\frac{1}{3}p^2\\
&\geq (p-1)p-\frac{1}{3}p^2\\
&= \frac{2}{3}p^2-p \text{ (symbols per stripe).}
\end{aligned}$$

Thus, R is lower bounded by $(\frac{2}{3}p^2 - p)$ symbols. □

We use simulations to compare both conventional and replace recoveries with the lower bound. Figure 2 shows the results. We observe that replace recovery is very close to the lower bound for $p < 40$. We also verify that replace recovery can give a result lower than $0.69p^2$ for $p < 1000$.

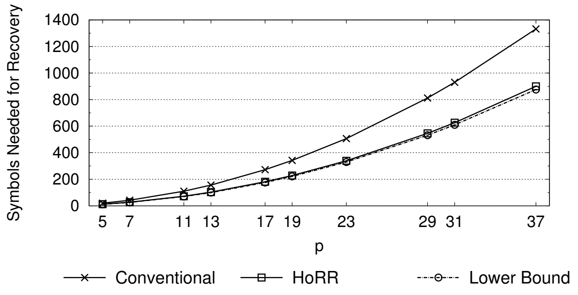
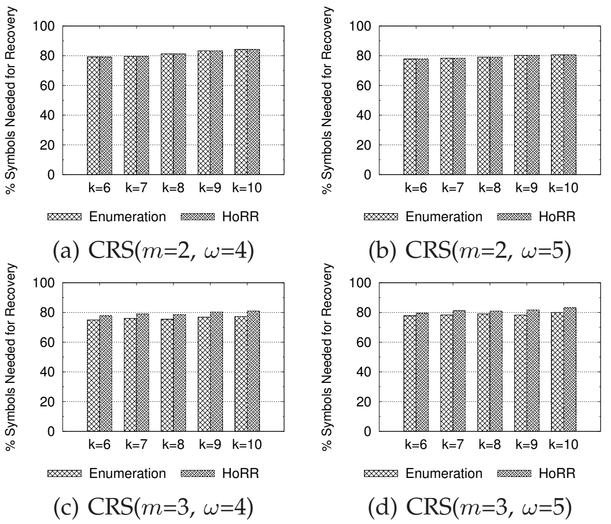


Fig. 2. Number of symbols per stripe needed for a single-node failure recovery in STAR.

CRS codes. We now consider CRS [2]. We use simulations to evaluate the savings of replace recovery over conventional recovery in terms of the number of read symbols. We also evaluate the savings of enumeration recovery to verify the accuracy of replace recovery. Figure 3 presents the results for different combinations of m, ω, and k. For $m = 2$, we observe that replace recovery achieves the same optimal result as in enumeration recovery, and the savings over conventional recovery are 15.75-25.00%. For $m = 3$, replace recovery achieves near-optimal performance. When compared to conventional recovery, the savings of replace recovery are 16.25-22.22%, while the savings of enumeration recovery are 19.75-25.00%.



(a) CRS(m=2, ω=4) (b) CRS(m=2, ω=5)

(c) CRS(m=3, ω=4) (d) CRS(m=3, ω=5)

Fig. 3. Number of symbols (per stripe) needed for single-node failure recovery for enumeration and HoRR recoveries in CRS with m=2 and m=3.

4 Implementation Details

In this section, we present the design and implementation for our replace recovery algorithm. We implement it on a parallel architecture that can scale the recovery performance in practice.

4.1 Recovery Thread

We implement the recovery operation with a *recovery thread*, a process that interconnects all nodes and coordinates all data reads and writes with the nodes. A recovery thread can be viewed as an intermediate controller process that relays data among the nodes. To recover a single-node failure, the recovery thread performs three steps: (i) reading data from the surviving nodes, (ii) reconstructing the lost data, and (iii) writing the reconstructed data to a new node. Note that the recovery thread implementation only requires the nodes support standard read/write functions.

4.2 Parallel Recovery Architecture

For further performance improvements, we implement a *parallel recovery architecture* that parallelizes the recovery operation via multi-threaded and multi-server designs. Figure 4 shows the architecture.

Multi-threaded recovery. As modern architectures shift toward multi-core, parallelizing the recovery process with multiple recovery threads becomes possible within a multi-core server. Two multi-threaded techniques have been proposed for recovery in RAID systems [8]: *disk-oriented reconstruction (DOR)* and *stripe-oriented reconstruction (SOR)*. We can implement the DOR and SOR as follows. In DOR, we create $n > 1$ recovery threads, each associated with one node, where n is the total number of nodes. Each of the $n - 1$ threads reads data chunks from its associated surviving node, and the remaining thread reconstructs and writes the resulting data chunks to the new node. In contrast, in SOR, we create multiple recovery threads, each associated with a group of stripes (note that each stripe spans all the nodes). Since each stripe is independently encoded, each thread can recover its own group of stripes independently as well. Since DOR needs to manage many threads if the number of nodes increases, in our implementation, we use SOR as our multi-threaded recovery strategy.

Multi-server recovery. To further boost the recovery performance, we deploy a cluster of recovery servers to extend the scale of parallelism. The cluster is composed of one *dispatcher* and multiple *executers*, as shown in Figure 4. The dispatcher splits the whole recovery operation into different independent *tasks*, each of which corresponds to the recovery of a group of stripes. It then assigns each task to a different executer. An executer notifies the dispatcher after completing its task, so that it can be assigned the next task. Each assigned task can be further decomposed into different groups of stripes, each

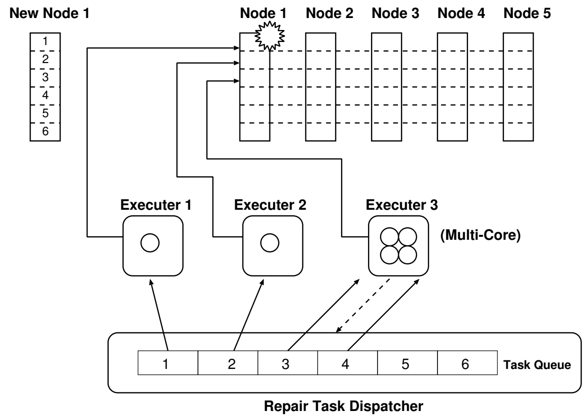
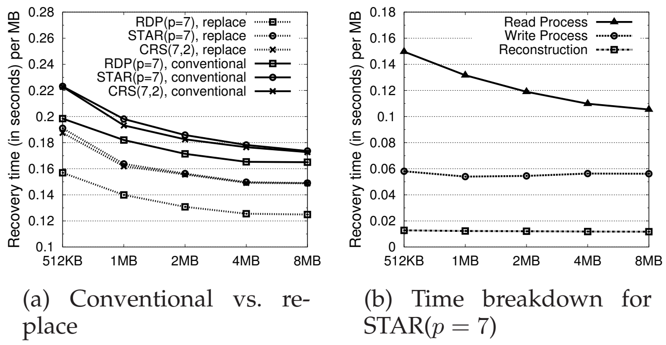


Fig. 4. Our parallel recovery architecture for scaling the recovery performance.



(a) Conventional vs. replace

(b) Time breakdown for STAR($p = 7$)

Fig. 5. Experiment 1 - Impact of chunk size.

being processed by a recovery thread based on the SOR approach. Note that in actual deployment, the executors are deployed in different servers, while the dispatcher may be deployed in one of the executor servers, given that its dispatching workload is lightweight in general.

4.3 Implementation of XOR-based Erasure Codes

We implement both conventional and replace recoveries for the following XOR-based erasure codes: RDP [23], EVENODD [21], X-code [24] (all of which are double-fault tolerant), STAR (triple-fault tolerant), and Cauchy Reed-Solomon (CRS) codes (multi-failure tolerant). While there are many existing double-fault tolerant XOR-based erasure codes, we pick RDP, EVENODD, and X-Code because their optimal recovery solutions have been found (see [21], [23], [24], respectively) and we can compare the solutions of our replace recovery approach with their respective optimal results. Note that the numbers of nodes (i.e., n) being used for RDP, EVENODD, X-Code, and STAR are mainly determined by a prime number p. For clarity, Table 1 summarizes the configurations of the codes that we have implemented based on p.

TABLE 1
Configurations of the codes that we consider.

Code	n	k	m	Remarks
RDP	$p+1$	$p-1$	2	prime $p > 2$
EVENODD	$p+2$	p	2	prime p
X-Code	p	$p-2$	2	prime p
STAR	$p+3$	p	3	prime p
CRS	For general $n = k + m$			

We set the unit of XOR of encoding/decoding to be four bytes long, so as to make our implementation compatible with both 32-bit/64-bit machines [18]. The stripe unit in CRS is a ω-bit word, where ω must be large enough so that the total number of nodes n is at most 2^{ω}. Note that in CRS, ω does not need to be a multiple of the machine word length, but should be as small as possible. Here, we select $\omega = 6$ for CRS, meaning that we allow at most 64 nodes in the system. Note that from our simulations, we also see the improvements of our replace recovery algorithm for different values of ω (see Figure 3).

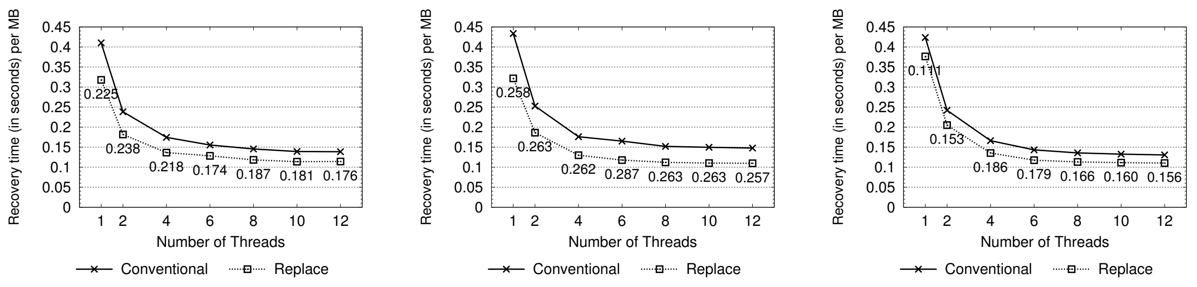
In our implementation, we treat each symbol as a chunk, which can be of large size in general. Unlike typical file systems that use small block sizes (for example, the default block size in Linux file systems is 4KB), operating on large chunks is typical in distributed storage systems (e.g., GFS [7] uses the chunk size 64MB). We evaluate how the recovery performance is influenced by the chunk size in our experiments in the next section.

5 Additional Experiments

Experiment 1: Impact of chunk size. We first evaluate how different chunk sizes influence the recovery performance using HoRR. We consider different chunk sizes, ranging from 512KB to 8MB. We focus on various coding schemes that can tolerate different numbers of failures, including RDP($p = 7$), STAR($p = 7$), and CRS($k = 7$, $m = 2$).

Figures 5(a) shows the recovery time (per MB) for different chunk sizes using conventional recovery and replace recovery. We observe that as the chunk size increases, the recovery time decreases. The reason is that given the same amount of data, the number of accesses decreases for a larger chunk size. The rate of decrease diminishes as the chunk size further increases, so we expect that the recovery time stabilizes for a large enough chunk size. As shown in the figure, the decrease trend applies to both conventional and replace recoveries. In fact, similar results are observed for all coding schemes and this effect is also demonstrated in prior work [11].

As described in Section 4.1, a recovery operation consists of three main parts. In order to evaluate the contribution of each part to the whole recovery performance, we provide a performance breakdown for the recovery operation. Here we take conventional recovery for STAR($p = 7$) as an example. Figure 5(b) shows the breakdown (i.e., reading data from surviving nodes, reconstructing lost data, and writing data to a new node) for STAR. We observe that the reconstruction part contributes less than 10% in STAR, and we believe that the



(a) RDP($p = 13$) (b) STAR($p = 13$) (c) CRS($k = 12$, $m = 4$)

Fig. 6. Experiment 2 - Multi-threaded recovery based on SOR.

XOR-based encoding/decoding operations in STAR have minimal computational overhead. More importantly, the read part contributes over 60% of the overall recovery time for all chunk sizes. Note that if the number of nodes increases, then more data will be read from surviving nodes, so it is expected that the read part will contribute a larger proportion to the overall recovery time. To summarize, the experiment results show that in order to reduce the overall recovery time, it is critical to minimize the number of read symbols, and hence the amount of data read from surviving nodes.

In our experiments, we fix the default chunk size to be 512KB, which been chosen by some existing storage systems (e.g., OBFS [20]). Although the 512KB chunk size gives the maximum (worst) recovery time in general, our main goal is to compare the relative recovery performance of conventional and replace recoveries, rather than their actual recovery performance.

Experiment 2: Parallel recovery. We now evaluate the recovery performance of HoRR based on parallelization (see Section 4.2). We aim to show that replace recovery still outperforms conventional recovery in parallelized implementation. Here, we use RDP($p = 13$), STAR($p = 13$), and CRS($k = 12$, $m = 4$) as representatives for different degrees of fault tolerance.

We first evaluate the recovery performance of SOR-based multi-threaded implementation, while we still use a single recovery server. Figure 6 shows the recovery time versus the number of recovery threads being deployed in a single server. As the number of threads increases (with no more than four threads), the recovery times for both conventional and replace recoveries significantly decrease. For example, let us consider CRS($k = 12$, $m = 4$). When four recovery threads are used, the recovery times of conventional and replace recoveries are reduced by 61% and 64%, respectively when compared to the single-threaded case. However, when the number of threads goes beyond four, the improvement is marginal. The main reason is that the performance gain is bounded by the number of CPU cores (recall that our recovery server is equipped with a Quad-Core CPU). More importantly, the results presented in Figure 6 show the applicability of replace recovery in parallelized implementation. We observe that replace recovery uses less recovery time than conventional recovery *regardless of the number of recovery threads being used*. For example, in STAR, replace recovery reduces the recovery time of conventional recovery by 25.7-28.7%; in CRS, the recovery time reduction is 15.3-18.6% when more than one recovery thread is used.

We now evaluate the recovery performance when we use multiple recovery servers. Here, we deploy two executors in two separate Quad-Core servers (as opposed to one Quad-Core server in our prior experiments) for parallel recovery. We configure each executor server to run four recovery threads (i.e., we have a total of eight recovery threads). Also, we deploy the dispatcher in one of the executor servers. In multi-server recovery mode, the dispatcher splits the whole recovery process into eight tasks (with a group of stripes). It first dispatches one task to each of the two executers. When one of the executors finishes its assigned task, the dispatcher assigns that executor another task.

We now measure the recovery time performance with this parallel setup. Figure 7 compares the recovery times of four recovery approaches: (i) single-server, multi-threaded (conventional), (ii) single-server, multi-threaded (replace), and (iii) multi-server (conventional), and (iv) multi-server (replace). We observe that multi-server implementation reduces the recovery time compared to the single-server implementation. For example, for replace recovery, the multi-server implementation reduces the recovery time by 24.4%, 22.0%, 19.82% for RDP, STAR, and CRS, respectively when compared to the single-server approach. In theory, we should expect 50% reduction, but the coordination overhead between the dispatcher and executors may degrade the actual performance. Nevertheless, the multi-server approach can provide additional recovery time improvements.

Note that even in multi-server implementation, we still observe the improvements of replace recovery over conventional recovery, as the recovery time is reduced by 25.3%, 28.0%, 21.50% for RDP, STAR, and CRS, respectively. The results also validate the applicability of replace recovery in parallelized implementation.

References

[1] M. Blaum and R. Roth. On Lowest Density MDS codes. *Information Theory, IEEE Transactions on*, 45(1):46–59, 1999.

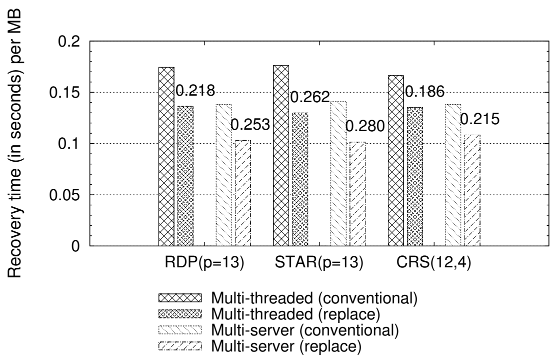


Fig. 7. Experiment 2 - Parallel recovery. Each number denotes the overall percentage decrease of recovery time compared to conventional recovery.

[2] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An XOR-Based Erasure-Resilient Coding Scheme. *International Computer Sciences Institute Technical Report ICSI TR-95-048*, 1995.

[3] DataRecovery. http://www.datarecovery.com/.

[4] DataTech Labs. http://www.datatechlab.com/.

[5] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran. Network Coding for Distributed Storage Systems. *IEEE Trans. on Information Theory*, 56(9):4539–4551, 2010.

[6] K. Esmaili, P. Lluis, and A. Datta. The CORE Storage Primitive: Cross-Object Redundancy for Efficient Data Repair & Access in Erasure Coded Storage. *arXiv*, preprint arXiv:1302.5192, 2013.

[7] S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. In *Proc. of ACM SOSP*, 2003.

[8] M. Holland, G. Gibson, and D. Siewiorek. Architectures and Algorithms for On-line Failure Recovery in Redundant Disk Arrays. *Distributed and Parallel Databases*, 2(3):295–335, 1994.

[9] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure Coding in Windows Azure Storage. In *Proc. of USENIX ATC*, Jun 2012.

[10] C. Huang and L. Xu. STAR: An Efficient Coding Scheme for Correcting Triple Storage Node Failures. *IEEE Trans. on Computers*, 57(7):889–901, 2008.

[11] O. Khan, R. Burns, J. S. Plank, W. Pierce, and C. Huang. Rethinking Erasure Codes for Cloud File Systems: Minimizing I/O for Recovery and Degraded Reads. In *Proc. of USENIX FAST*, 2012.

[12] J. Li, S. Yang, and X. Wang. Building Parallel Regeneration Trees in Distributed Storage Systems with Asymmetric Links. In *Proc. of CollaborateCom*, pages 1–10. IEEE, 2010.

[13] J. Li, S. Yang, X. Wang, and B. Li. Tree-Structured Data Regeneration in Distributed Storage Systems with Regenerating Codes. In *Proc. of IEEE INFOCOM*, 2010.

[14] D. Papailiopoulos, J. Luo, A. Dimakis, C. Huang, and J. Li. Simple Regenerating Codes: Network Coding for Cloud Storage. In *Proc. of IEEE INFOCOM*, Mar 2012.

[15] J. Plank. A New Minimum Density RAID-6 Code with A Word Size of Eight. In *Network Computing and Applications, 2008. NCA'08. Seventh IEEE International Symposium on*, pages 85–92. IEEE, 2008.

[16] J. Plank. The RAID-6 Liberation Codes. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, page 7. USENIX Association, 2008.

[17] J. Plank, A. Buchsbaum, and B. Vander Zanden. Minimum Density RAID-6 Codes. *ACM Transactions on Storage (TOS)*, 6(4):16, 2011.

[18] J. Plank, J. Luo, C. Schuman, L. Xu, and Z. Wilcox-O'Hearn. A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries for Storage. In *Proc. of USENIX FAST*, pages 253–265, 2009.

[19] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur. XORing Elephants: Novel Erasure Codes for Big Data. *Proceedings of the VLDB Endowment*, 2013.

[20] F. Wang, S. Brandt, E. Miller, and D. Long. OBFS: A File System for Object-based Storage Devices. In *Proc. of IEEE MSST*, pages 283–300. Citeseer, 2004.

[21] Z. Wang, A. Dimakis, and J. Bruck. Rebuilding for Array Codes in Distributed Storage Systems. In *IEEE GLOBECOM Workshops*, 2010.

[22] C. Wu, X. He, G. Wu, S. Wan, X. Liu, Q. Cao, and C. Xie. HDP Code: A Horizontal-Diagonal Parity Code to Optimize I/O Load Balancing in RAID-6. In *Proc. of IEEE/IFIP DSN*, 2011.

[23] L. Xiang, Y. Xu, J. Lui, Q. Chang, Y. Pan, and R. Li. A Hybrid Approach to Failed Disk Recovery Using RAID-6 Codes: Algorithms and Performance Evaluation. *ACM Trans. on Storage*, 7(3):11, 2011.

[24] S. Xu, R. Li, P. Lee, Y. Zhu, L. Xiang, Y. Xu, and J. Lui. Single Disk Failure Recovery for X-Code-Based Parallel Storage Systems. *IEEE Trans on Computers*, 2013. (To appear).

[25] Y. Zhu, P. P. C. Lee, L. Xiang, Y. Xu, and L. Gao. A Cost-based Heterogeneous Recovery Scheme for Distributed Storage Systems with RAID-6 Codes. In *Proc. of IEEE/IFIP DSN*, 2012.