Erasure Code with Shingled Local Parity Groups for Efficient Recovery from Multiple Disk Failures

Takeshi Miyamae Takanori Nakao Kensuke Shiozawa
{miyamae.takeshi, nakao.takanori, shiozawa.kennsu}@jp.fujitsu.com
Fujitsu Laboratories Ltd.

Abstract

The ever-growing importance and volume of digital content generated by ICT services has led to the demand for highly durable and space-efficient content storage technology. Erasure code can be an effective solution to such requirements, but the current research outcomes do not efficiently handle simultaneous multiple disk failures. We propose Shingled Erasure Code (SHEC), an erasure code with local parity groups shingled with each other, to provide efficient recovery for multiple disk failures while ensuring that the conflicting properties of space efficiency and durability are adjustable according to user requirements. We have confirmed that SHEC meets the design goals using the result of a numerical study on the relationships among the conflicting properties, and a performance evaluation of an actual SHEC implementation on Ceph, a type of open source scalable object storage software.

1. INTRODUCTION

The ever-growing importance of digital content generated by ICT service vendors has led to an increasing demand for highly durable content storage technology. For ICT services, triple replication technique has met this demand for years, but its low space efficiency (three times the storage capacity of user data) has made this solution less attractive for vendors. Erasure codes are a durable data storage technique with less redundant information (parities), and are rapidly gaining popularity.

The space efficiency and durability of erasure codes come with increasing computational overhead. In particular, the recovery overhead associated with disk failures severely affects the availability and performance of ICT services unless the performance interference of the recovery operation is controlled properly. Throttling the CPU and I/O bandwidth of the recovery operation to minimize interference is one of the common practices [18], but controlled interference entails a longer recovery operation, which, in turn, makes the storage less durable. Hence recovery performance enhancement itself has been a focus of research in erasure code field lately [10].

Recent, remarkable achievements in recovery performance enhancement are Microsoft LRC (MS-LRC) [9] and Facebook Xorbas [10]. They use the concept of local parity, where a few instances of parity are calculated based only on subsets of an entire dataset. Given that the amount of data read and transferred is reduced only in those subsets of a dataset, the recovery consumes a smaller amount of each ICT resource and becomes faster.

However, in the event of multiple disk failures, MS-LRC and Xorbas use global parities, which include parity information calculated over an entire dataset, and a large amount of data is transferred in the recovery operation. Because each of the disk failures is often correlated in the real world [16], we are motivated to develop a new erasure code that is robust against multiple disk failures.

In this paper, we propose Shingled Erasure Code (SHEC), whose local parity groups overlap each other. The code is designed to recover efficiently from multiple disk failures, and space efficiency and durability are user-adjustable. To confirm the design goals of SHEC, we first study in section 2 the relationships among the three key properties of erasure codes (space efficiency, durability and recovery efficiency) with our various local parity group layouts. Then, we demonstrate that the layout is adjustable to achieve the optimal combination of the intended properties to users. In section 3, we show the aforementioned recovery efficiency of SHEC through evaluation with the SHEC implementation in Ceph[14], a type of open source scalable object storage software.

2. ANALYSES OF LOCAL PARITY GROUPS AND A NEW CODE

2.1. Properties of Local Parity Groups

In this paper, we call each split data element in an erasure code stripe a 'data chunk' and each of the local parity groups a 'parity chunk.' We define the local parity's locality as the number of data chunks to calculate each parity chunk.

We pick three main properties in the erasure code with local parity groups.

- Space efficiency
- Durability
- Recovery efficiency

The first property, space efficiency, is defined as the ratio of data chunks and calculated as k/(k+m) (k: the number of data chunks, m: the number of parity chunks), indicating cost efficiency.

The second property, durability, is defined as the probability of data loss, as shown in equation (1) [17].

$$PDL =_n P_f \times (1/MTBF) \times (MTTR/MTBF)^{f-1} \quad (1)$$

PDL: Probability of data loss
MTTR: Mean time to recovery
MTBF: Mean time between failures
n: Total number of disks
f: Number of concurrent disk failures
${}_nP_f$: n!/(n-f)!

The last property, recovery efficiency, is defined as the inverse number of recovery overhead (recovery overhead means the ratio of read chunks to all data chunks during the recovery). Generally, the smaller the locality is, the higher the recovery efficiency is. The recovery efficiency raises the availability or performance of ICT services.

Next, we now explain the three-way trade-off relationship among the abovementioned erasure code properties, as shown in Figure 1. First, we mention the relationship between space efficiency and durability. This trade-off is obvious because if we add parities, durability increases whereas space efficiency decreases (trade-off #1). The second relationship is between durability and recovery efficiency. If we reduce the locality to increase recovery efficiency, the number of parity chunks covering each data chunk decreases, indicating a decrease in durability (trade-off #2). The last relationship is between space efficiency and recovery efficiency. To reduce the locality with equal durability, we must add more local parities to keep the number of parity chunks covering each data chunk, indicating a decrease in space efficiency (trade-off #3).

Customers often opine that durability should not be sacrificed to increase recovery efficiency. In such a case, we usually suggest sacrificing space efficiency instead of durability because space efficiency and recovery efficiency also share a trade-off relationship (#3).

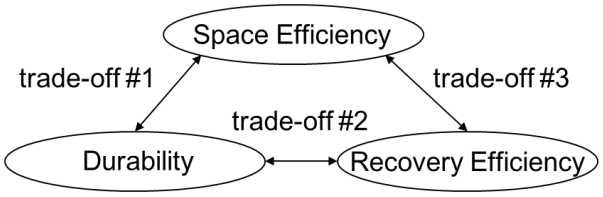


Figure 1: Three-Way Trade-Off

In the remainder of this paper, we will propose and evaluate the new erasure code with local parity groups, based on the above analyses.

2.2. Shingled Erasure Code

We propose a new erasure code, Shingled Erasure Code (SHEC), which is designed for efficient recovery in the event of multiple disk failures, with space efficiency and durability adjustable according to user requirements. SHEC is an erasure code with local parity groups, and the calculation ranges of local parities are shifted and partly overlap with each other, similar to arranging shingles on the roof of a house. All local parity groups have the same locality and are shifted at almost regular intervals. SHEC(k,m,l) represents a layout with k data chunks, m parity chunks and locality l.

The average number of parity chunks that have relation to each data chunk is ml/k. Because the failure of ml/k+1 data or parity chunks can cause data loss, we use ml/k as an estimator of SHEC's durability. For example, in the case of SHEC(10,6,5), the estimator is 3 (= ml/k = 6*5/10) and the four failures of D1/P1/P5/P6 cause data loss because D1 cannot be recovered from the remaining chunks (Figure 2).

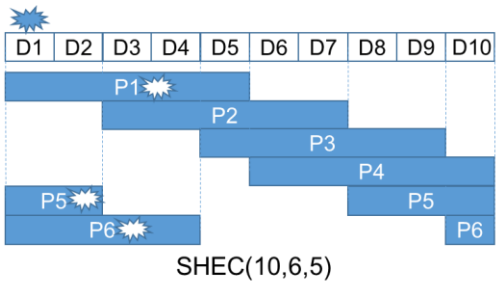


Figure 2: Estimation of SHEC's Durability

In the case of multiple data chunk failures, SHEC recovers data from multiple parity chunks. Usually, there are multiple combinations of parity chunks that recover the failed chunks. SHEC selects the one which requires the lowest number of disks read. For example, when D6/D9 fail in Figure 2, SHEC selects P3/P4 because the union of P3/P4's calculation ranges results in six contiguous data chunks, and the size of the union (that indi-

cates the amount of data read) is the least among all candidate parity chunk pairs.

To show SHEC's improvement factor in durability, we compare SHEC(6,4,3) with an instance of simple local parity groups (SLPG) (Figure 3). If D1/D2/D3 fail simultaneously, the SLPG cannot be recovered because only P1/P3 have relation to D1/D2/D3. In contrast, SHEC(6,4,3) can be recovered from P1/P2/P4. Among all failure patterns, SHEC(6,4,3)'s data loss cases number half those of the SLPG. That means that SHEC(6,4,3)'s durability is twice as high as SLPG with the same space efficiency and recovery efficiency. Moreover, the higher the durability estimator is, the more the improvement factor of durability is (Figure 4). The improvement is ascribed to the shifting of the calculation range for each local parity.

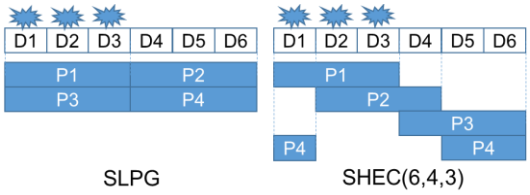


Figure 3: SLPG vs. SHEC

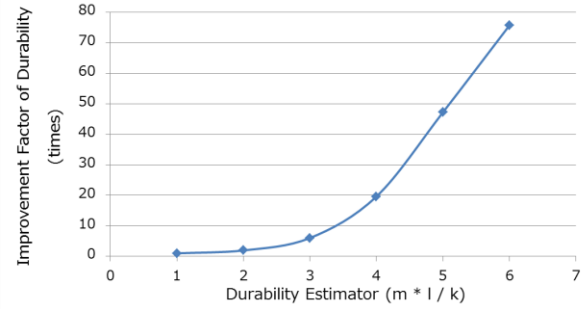


Figure 4: SHEC's Improvement Factor of Durability

Formulating the SHEC generator matrix is quite simple. First, we create a generator matrix of Reed Solomon systematic code (abbreviated as RS(k,m) in this paper). Next, each matrix element whose corresponding data chunk is not used for calculating the corresponding parity chunk is set to zero (Figure 5). CPU utilization is directly proportional to the number of non-zero matrix elements.

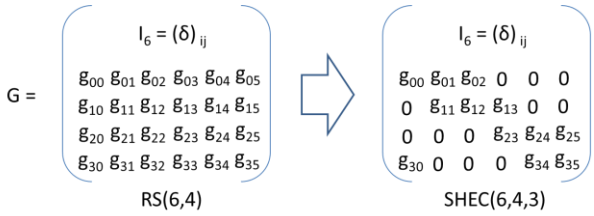


Figure 5: SHEC Generator Matrix

2.3. Comparison among SHEC Parameter Sets

In this section, we show that SHEC provides layouts with less recovery overhead, ensuring that conflicting properties of space efficiency and durability are adjustable according to user requirements.

Restricted to the meaningful ones, SHEC can generate over 100 different parameter sets in the three-dimensional property space. We pick some sets from them and compare their erasure-code properties. Let us start with SHEC(4,2,4), RAID6's equivalent SHEC parameter set, and search for alternative candidates that are more recovery-efficient with almost equal durability (Table 1). In this case, we sacrifice the space efficiency, and we can get the candidates SHEC(4,3,3) and SHEC(6,4,3).

k	m	l	ml/k	Space Effic.	Durability (Annual)	Rcvr-Ovhd (1x/2x fail)
4	2	4	2	67%	1.44E-17	1.00/1.00
4	3	3	2.25	57%	1.60E-18	0.75/1.00
6	4	3	2	60%	3.46E-18	0.50/0.74

Table 1: Candidates with Equal Durability

Let us search for other candidates that are more recovery-efficient with almost equal space efficiency (Table 2). In this case, we sacrifice the durability, and we can get the candidates SHEC(5,3,3) and SHEC(7,4,3).

k	m	l	ml/k	Space Effic.	Durability (Annual)	Rcvr-Ovhd (1x/2x fail)
4	2	4	2	67%	1.44E-17	1.00/1.00
5	3	3	1.8	63%	1.22E-10	0.60/0.90
7	4	3	1.71	64%	1.65E-10	0.43/0.69

Table 2: Candidates with Equal Space Efficiency

2.4. Comparison with Other Erasure Codes

We compare the theoretical recovery overhead between the Reed Solomon code, MS-LRC, Xorbas and SHEC, under the condition of almost equal durability (Figure 6). SHEC's recovery overhead is less than the others in cases of double or more disk failures. The others are worse because they must use global parities in those cases.

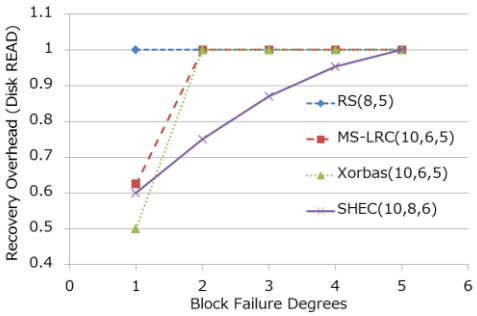


Figure 6: Recovery Overhead between Codes

3. IMPLEMENTATION AND EVALUATION

3.1. Ceph Architecture

We evaluated SHEC on Ceph. A Ceph cluster includes a large number of object storage daemons (OSDs). Each OSD corresponds to a storage device. In this paper, an OSD indicates a whole disk device. A placement group (PG) is a set of OSDs over which the data and parity chunks are distributed randomly.

When a data is written on a Ceph cluster, (a) the data is divided into 4MB Ceph objects. Next, (b) the PG is determined by a hash value for the name of the object. Finally, (c) each item of data or parity chunk is stored in one of the OSDs assigned to the PG. When (d) an OSD fails, (e) another OSD is newly assigned to the degraded PG, and the lost chunk is recovered to the OSD from the remaining chunks in the acting-set.

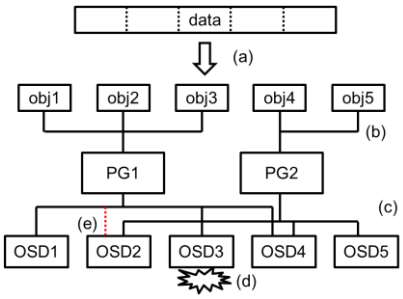


Figure 7: Ceph Architecture

3.2. SHEC Implementation

The Ceph object storage has an interface plugin for erasure code from v0.80.1 (Firefly) release. It supported the Reed Solomon code as a default plugin, and we implemented SHEC as an alternative. The interface is sufficient to implement SHEC because it includes a useful function minimum_to_decode(), which yields the set of chunk numbers required to decode the lost chunks. SHEC yields the subset of the chunk numbers that the Reed Solomon code yields.

3.3. Test Conditions

We evaluated the SHEC(6,4,3)'s recovery performance and selected RS(6,4) as a reference (Figure 8). The comparison with Xorbas or MS-LRC is beyond the scope of this paper.

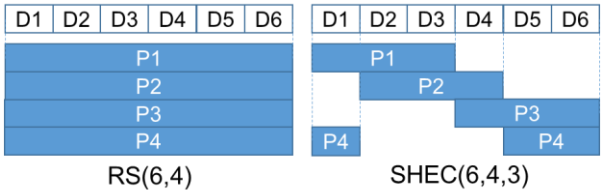
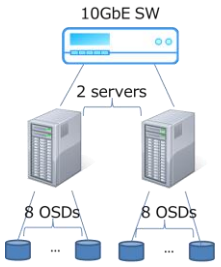


Figure 8: Parity Layouts for Comparison

Hardware setup and software version for testing is described in Figure 9.



Machine	Fujitsu RX300 S7 (x2)
CPU	Xeon 300GHz (8 core)
Memory	32GB
HDD	300GB SAS (x8)
Network	10GbE NIC
OS	Ubuntu 14.04
Kernel	3.13.0-24
Ceph	v0.80.1 (Firefly)

Figure 9: Hardware Setup and Software Version

3.4. SHEC Recovery Performance

We prepared 100GB (25,000 * 4MB objects) of data and measured the recovery time from double OSD failures in each of the cases of RS(6,4) and SHEC(6,4,3). We set the number of Ceph recovery threads as five. At first, we found that the SHEC(6,4,3)'s total CPU overhead was about 20% less than that of RS(6,4) as shown in Figure 10. We consider that this was mainly due to the effect of our simplified generator matrix (Figure 5) because CPU utilization is directly proportional to the number of non-zero matrix elements.

Next, RS(6,4)'s recovery overhead is 1.00 (the number of read chunks is equal to k), whereas that of SHEC(6,4,3) is 0.74 (Table 1). Therefore, we estimated SHEC(6,4,3)'s recovery time at 74% of RS(6,4) in the beginning. However, this experiment showed that, though the amount of data read from the disks was 74%, the actual recovery time was 81.4% of RS(6,4) (Figure 10).

The reason was a partial bottleneck. We assumed that one of the system resources must be bottlenecked, and in fact, the disk seemed bottlenecked (Figure 11), while the CPU and network (Figure 10, Figure 12) did not at all. However, the disk bottleneck did not continue constantly. Seeing Figure 11, the disk bandwidth was not fully utilized during 35% of entire recovery time (in the rectangle), and we could re-estimate SHEC(6,4,3)'s recovery time as follows.

$$0.74 * (1-0.35) + 1.0 * 0.35 = 0.831 \qquad (2)$$

The result was 83% of RS(6,4), almost the same as the actual ratio, 81.4%.

Finally, we concluded that SHEC's recovery overhead was decreased in comparison with the Reed Solomon code. However, in our test conditions, 70% of SHEC's recovery efficiency emerges as decreasing latency of recovery completion, and 30% emerges as decreasing disk bandwidth used for recovery processing.

Moreover, we obtained a similar result when we tried the comparison between RS(5,3) and SHEC(5,3,3) in the event of single OSD failure.

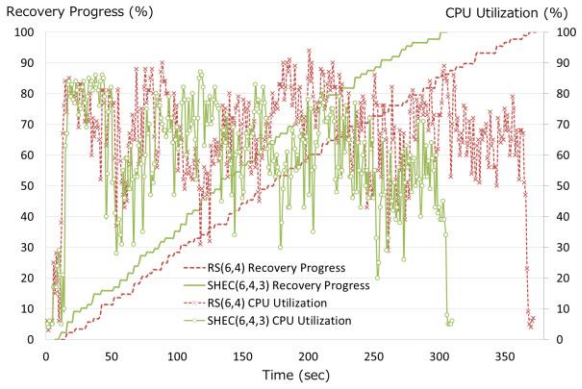


Figure 10: Recovery Progress and CPU Utilization

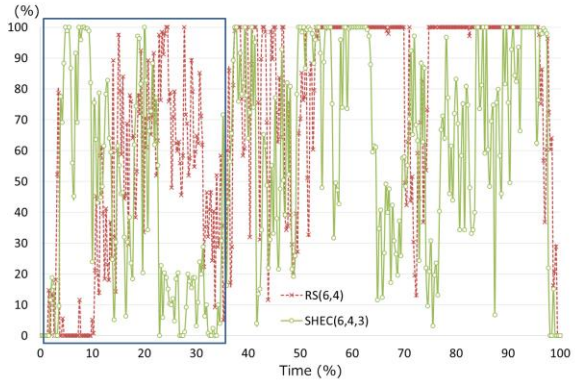


Figure 11: Disk Utilization

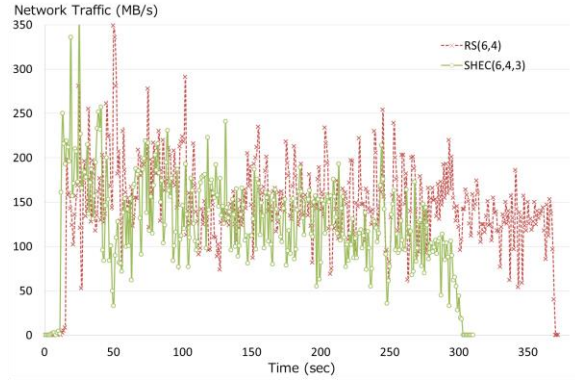


Figure 12: Network Traffic

4. RELATED WORK

Various storage systems have used erasure codes [1] to realize a higher durability of storage data for a long time. Especially, the well-known RAID [1], based on the Reed Solomon code, was deployed in almost all of the highly reliable storage systems.

After Google disclosed Google File System [3], which used triple replication with the diffusion of commodity disks, other major cloud storages such as Apache HDFS [4] and Microsoft Azure storage [5] followed the trend. However, because the volume of data generated by ICT services started to grow explosively, the erasure code's space efficiency was revalued again [6].

In recent years, recovery overhead has become regarded as a serious problem of erasure code, especially in distributed or scalable storages [9], [10]. Many researchers have proposed methods including local parity techniques to decrease the recovery overhead. WEAVER Codes [7] suggested a generic method which includes most of the possible local parity layouts. Microsoft Pyramid Codes [8], followed by Azure's Local Reconstruction Codes (MS-LRC) [9] and Facebook Xorbas [10], discussed the durability of local parities. MS-LRC insisted that the probability of data loss (information-theoretically non-decodable case) is limited to the trivial level. On the other hand, Xorbas discussed the relationship between locality and code distance highly theoretically. Regenerating Codes [11], [12] suggested an interesting approach which discussed the properties of the optimal trade-off between space efficiency and recovery bandwidth via 'cut-based' analysis. Rotated Reed-Solomon Codes [13] suggested local parities and has similar layouts to SHEC. However, it is characterized by the number of data disks being limited to the product of a pair of integers. Fountain code [19] also seems to use a sliding window, but takes a highly probabilistic approach.

5. CONCLUSION AND FUTURE WORK

First, we proposed SHEC, a new erasure code designed for high recovery efficiency, especially from multiple disk failures. Second, we showed that SHEC provides the layouts with more recovery efficiency than the Reed Solomon codes, ensuring that the conflicting properties of space efficiency and durability are adjustable according to user requirements. Finally, we showed through experiments that the SHEC's recovery is actually faster than the Reed Solomon codes.

However, it may not be easy to show the normal SHEC's superiority to state-of-the-art codes (Xorbas and MS-LRC) without any sacrifices of space efficiency or durability. Therefore, we will expand the normal SHEC concept into an asymmetric one or one bundled with global parities in the future.

6. REFERENCES

[1] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *Computer Communication Review*, April 1997.

[2] David A. Patterson, Garth A. Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, pages 109-116, Chicago, Illinois, September 1988.

[3] Ghemawat, S., Gobioff, H., and Leung, S.-T. The Google File System. In *19th Symposium on Operating Systems Principles*. Lake George, NY. 29-43, December 2003.

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1-10, Washington, DC, USA, 2010. IEEE Computer Society

[5] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. Fahim ul Haq, M. Ikram ul Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas. Windows Azure storage: A highly available cloud storage service with strong consistency. In *Symposium on Operating Systems Principles*, 2011.

[6] H. Weatherspoon and J. D. Kubiatowicz. Erasure coding vs. replication: A quantitiative comparison. In *Procs. of IPTPS*, 2002.

[7] J. L. Hafner. WEAVER Codes: Highly fault tolerant erasure codes for storage systems. *FAST 2005*, San Francisco, CA, Dec. 2005.

[8] C. Huang, M. Chen, and J. Li. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. Presented at the *IEEE Int. Symp. Network Computing and Applications*, Jul. 2007.

[9] C. Huang, H. Simitci, Y. Xu, A.Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in Windows Azure Storage. Presented at the *USENIX Annu. Tech. Conf.*, Boston, MA, 2012.

[10] M. Sathiamoorthy, M. Asteris, D. S. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, D. Borthakur. XORing elephants: novel erasure codes for big data. In *Proc. of the Very Large Data Bases conference (VLDB)*, 2013, pp. 325-336.

[11] Y. Wu, A. G. Dimakis, and K. Ramchandran. Deterministic regenerating codes for distributed storage. Presented at the *Allerton Con. Control, Computing, and Communication*, Urbana-Champaign, IL, Sep. 2007.

[12] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran. Explicit and optimal exact-regenerating codes for the minimum-bandwidth point in distributed storage. In *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Austin, Jun. 2010, pp. 1938-1942.

[13] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang. Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads. In *FAST 2012*.

[14] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proc. of the 7th Symposium on Operating Systems Design and Implementation*, Seattle, WA, November 2006.

[15] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn. CRUSH: Controlled, scalable, decentralized placement of replicated data. In *Proc. Supercomputing (SC)*, 2006.

[16] D. Ford, F. Labelle, F. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *OSDI*, 2010.

[17] Richard Elling. A story of two MTTDL models. Ramblings from Richard's Ranch, 2007. Retrieved from https://blogs.oracle.com/relling/entry/a_story_of_two_mttdl

[18] M. Silberstein, L. Ganesh, Y. Wang, L. Alvisi and M. Dahlin. Lazy means smart: Reducing repair bandwidth costs in erasure-coded distributed storage. In *Proc. SYSTOR*, Jun. 2014.

[19] M. Asteris and A. G. Dimakis. Repairable fountain codes. In *Proc. Int. Symp. Inform. Theory*, Cambridge, MA, July 2012, pp. 1752–1756.