Bundling Together RAID Disk Arrays for Greater Protection and Easier Repairs

Jehan-François Pâris
Department of Computer Science
University of Houston
Houston, TX 77204-3010
jfparis@uh.edu

***Abstract*—Large data storage systems often use Reed-Solomon erasure codes to protect their static data against triple or even quadruple disk failures. The main drawback of this solution is the high cost of recovering the contents of failed disks, as it requires reading the contents of all the surviving peers in its parity stripe. We propose to reduce this cost by grouping together a small number of conventional RAID arrays into a single bundle and adding column-wise parity disks such that all disks in a given RAID array belong to a distinct parity stripe. As a result, the number of disks involved in the recovery from a single disk failure will be equal to the number of RAID arrays in the bundle. We show that bundles of RAID level 5 arrays can recover without data loss from all triple and at least 96 percent of quadruple disk failures while bundles of RAID level 6 arrays can similarly recover from all quintuple and 99.9 percent of sextuple disk failures.**

Keywords— Data storage systems, fault tolerant systems, parity check codes

I. Introduction

Large data storage systems have reliability requirements that cannot be met by either mirroring or by conventional RAID level 6 arrays [2] [14]. For instance, both the Google file system [4] and Windows Azure Storage [3] maintain three replicas of their active data. The sole disadvantage of this approach is its low space efficiency as two-thirds of the storage space contains redundant data. The preferred way to reduce this overhead is to use Reed-Solomon erasure codes [12] to store static or archival data. For instance, the BackBlaze cloud backup service [1] splits every incoming file into 17 equal-size shards and calculates 3 parity shards so that the file contents can be reconstituted from any 17 of these 20 shards. The main advantage of this approach is its low space overhead for the protection it offers. Even though parity information only occupies 15 percent of the disk space, the array can tolerate all triple disk failures without losing any data.

The sole disadvantage of this organization is the high cost of data recovery operations after the loss of a disk. Reconstituting the contents of the lost disk will require reading in the entire contents of 17 of the remaining 19 operational disks. That process is slow and is likely to take several hours. Transferring that amount of data will severely stress the communication layer of the storage system and slow down all other requests for the whole duration of the process. In addition, requests directed to the data not yet reconstituted will be excruciatingly slow, as each of them will result in 17 separate disk accesses.

This situation is acceptable in an online backup service as most of its data are likely to be never accessed. The same is not true for a distributed file system or a cloud storage service. This has motivated several coding solutions that address that issue. All these solutions trade a somewhat larger parity footprint for lowering the number of disks that must be accessed to reconstruct the contents of a failed disk. Two of these codes warrant our attention because they were developed for well-known large-scale storage solutions. These are the HDFS-XORBAS locally repairable code [13] and the Windows Azure local reconstruction code (LRC) [H12]. Both families of codes use modified Reed-Solomon erasure codes with additional local parity blocks supplementing the extant Reed-Solomon parity blocks.

A major limitation of these codes is their complexity. While their overall organization is intuitive, selecting the proper coding equations for the local parities is not a trivial task because suboptimal coefficient choices could significantly lower the reliability of the codes. In particular, any change in the number of data disks, local parity disks or global parity disks that a locally recoverable code manages requires a reevaluation of these coefficients.

We propose a much simpler approach that does not require any kind of tuning. We start by bundling together m regular RAID arrays. While we expect them to have equal sizes, this is not essential to our approach. We add to these arrays perpendicular RAID level 4 parity stripes in such a way that (a) each disk in the bundle belongs to one of the new parity stripe and (b) no two disks in any of the original RAID arrays belong to the same stripe. As a result, the bundle will be able to recover from any single disk failure by XORing the contents of exactly m disks.

Our scheme is flexible in several ways. First, we can vary the number of RAID arrays per bundle selecting different tradeoff points between recovery costs and space overhead. Second, we can arbitrarily select the sizes of the original RAID arrays. Finally, we can use single-parity or dual-parity arrays as constituting elements depending on the desired reliability level. We can indeed show that:

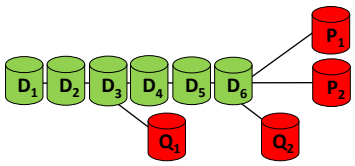

Fig. 1. A (6, 2, 2) Azure local reconstruction code.

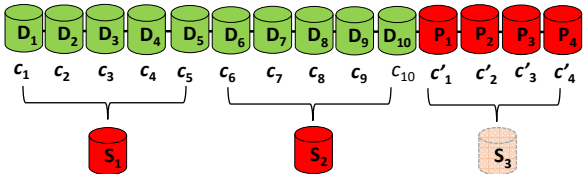

Fig. 2. A HDFS-XORBAS locally repairable code.

- Any reasonably sized bundle consisting of two or more single-parity RAID arrays will tolerate all triple disk failures and at least 96 percent of all quadruple failures.
- Any reasonably sized bundle consisting of two or more RAID level 6 arrays will tolerate all quintuple disk failures and at least 99.9 percent of all sextuple failures.

The remainder of the paper is structured as follows. Section II describes extant locally repairable codes. Section III introduces our proposal and Section IV discusses its main features. Finally, Section V has our conclusions.

II. Previous Work

In this section, we review the most significant previous work on locally repairable codes. Space considerations prevented us from being more complete.

A. Windows Azure Local Reconstruction Codes

Fig. 1 shows a (6, 2, 2) Azure Local Reconstruction Code (LRC) [6]. As we can see, the code partitions its six data blocks into two sets of three blocks with each set having its own local parity block, namely parity blocks Q_1 and Q_2. In addition, the code computes the two global parity blocks P_1 and P_2. More generally, a (k, l, r) code will partition its k data blocks into l sets of k/l blocks each and will have l local parity blocks and r global parity blocks. The code will be able to recover from the loss of either any single data block or any single local parity block using exactly k/l block reads. At the same time, the recovery performance of the code depends on the coefficients selected for the linear expressions defining the $l + r$ parity blocks. These coefficients must be specifically chosen to ensure that the code will be able to decode all the information theoretically decodable failure patterns. For a (6, 2, 2) LRC, this means all triple failures and 86 percent of all quadruple failures.

Overall, the (6, 2, 2) LRC ensures that all single data block failures can be resolved using three read operations and has a space overhead of 4/10 = 40 percent.

B. HDFS-XORBAS locally repairable codes

As we can see in Fig. 2, the HDFS-XORBAS locally repairable code [13] comprises 10 data blocks and 7 parity blocks. The four global parity blocks P_1,P_2, P_3, and $P_4$are built with a standard Reed-Solomon code and ensure that the code can tolerate the loss of four arbitrary blocks. Parity blocks S_1, S_2, and S_3 are local parity blocks aimed at reducing the cost of recovering from single block failures. Block S_1 is a linear combination $S = c_1D_1 \oplus c_2D_2 \oplus c_3D_3 \oplus c_4D_4 \oplus c_5D_5$ of the contents of data blocks D_1 to D_5 and block S_2 is similarly obtained from data blocks D_6 to D_{10}. Block S_3 is an *implied* parity block. It is not stored but can be created on demand as long as the coefficients of the linear expressions defining blocks S_1 and S_2 satisfy the relation $S_1 + S_2 + S_3 = 0$. In addition, the coefficients $c_i$are subjected to further optimization to maximize the fault tolerance of the code.

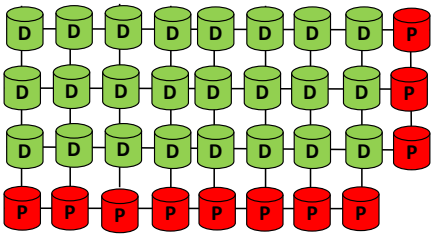

Fig. 3. A two-dimensional RAID array.

Overall, HDFS-XORBAS ensures that all single block failures can be resolved using five read operations and protects its contents against all quadruple disk failures. Its space overhead is 6/16 = 37.5 percent.

Comparing the (6, 2, 2) Azure LRC with HDFS XORBAS, we can see they make very different choices regarding the three-way tradeoff [8] among space efficiency, durability, and recovery efficiency. The (6, 2, 2) Azure LRC only requires three block reads to recover from a single data block failure but cannot tolerate all quadruple block failures. Conversely, HDFS XORBAS tolerates all quadruple block failures and has a somewhat smaller space overhead than the (6, 2, 2) Azure LRC (37.5 percent instead of 40 percent), but requires five read operations instead of three to recover from a single block failure.

C. Rotated Reed-Solomon codes

Khan et al. [7] investigated some of the most popular erasure codes and proposed a new class of codes that perform degraded reads more efficiently than all known codes, but otherwise keep the reliability and performance properties of extant Reed-Solomon codes. The emphasis of their work was on minimizing overall data transfers rather than minimizing the number of disks involved in the reconstruction.

D. Shingled Erasure Codes (SHEC)

Miyamae et al. [8] have proposed a disk array organization comprising k data disks and m parity disks. Each of these m parity disks contains the XOR of the contents of l data disks, which are said to form a *locality*. As $ml > k$, it is possible to assign each of the k data disks to exactly ml/k distinct localities in a way that ensures that the array will tolerate the simultaneous failure of up to ml/k disks. The organization is referred to as a shingled erasure code (SHEC) because localities overlap with each other like the tiles of a roof.

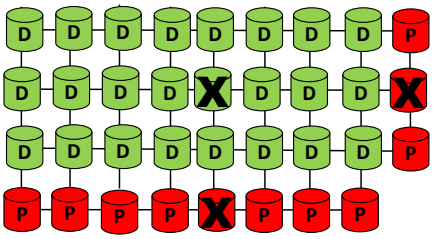

Fig. 4. One of the *mn* fatal triple failures.

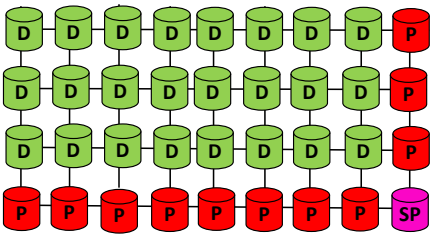

Fig. 5. Adding a superparity disk.

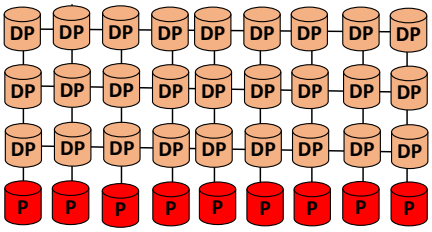

Fig. 6. A bundle of three RAID level 5 arrays protected by nine column parity blocks.

Shingled erasure codes guarantee that recovering from a single block failure will require exactly l block reads. Since all their parities are local, these codes can recover from double and triple failures without necessarily involving all the remaining data disks. In addition, these codes are easy to tune as their k and m parameters control the space overhead of the code $(k/(m+k))$ and the l parameter controls both its durability and its recovery efficiency. Low values of l will both guarantee a very efficient recovery from block failures and make the code less resilient to multiple block failures while high values of l will have the exact opposite effect.

E. Hitchhiker

Hitchhiker [11] is a modified Reed-Solomon code that reduces both network traffic and disk I/O by around 25 to 45 percent during reconstruction of missing data, without requiring any additional storage.

F. Rectangular RAID arrays

As Figure 3 shows, two-dimensional RAID arrays organize their data disks into *m* rows and *n* columns with a data disk at the intersection of each row and each column for a total of *mn* data disks. Each row contains a parity disk containing the parity of all the data disks in that row and each column contains a parity disk containing the parity of all the data disks in that column. As a result, the space overhead of the array is $(m+n)/(mn+m+n)$.

Consider now a rectangular RAID array having much fewer rows than columns ($m \ll n$). As Schwarz et al. [15] observed, the array would be able to recover from any single disk failure by computing the exclusive or of the remaining *m* disks of that column, without ever having to use row parities. At the same time, keeping *n* high allows us to control the parity overhead of the array. For instance, an array with three rows of ten data disks would only involve three disks in the recovery of any single data disk failure while only requiring a space overhead of $13/(30 + 13) = 30$ percent.

III. Our Proposal

A main problem with rectangular RAID arrays is that they do not provide the same level data protection as HDFS-XORBAS or the Azure LRC. While HDFS-XORBAS protects its contents against all quadruple disk failures and the Azure LRC can tolerate all triple failures without data loss, rectangular RAID arrays are vulnerable to triple failures involving a data disk and its two parity disks. Fig. 4 displays one of these fatal triple failures. The standard way to eliminate these triple failures is to add a superparity disk to the array [9] [16]. This superparity disk contains the XOR of either all the row or all the column parity disks. (The outcomes of the two operations would be identical.) The main drawback of this approach is that the superparity disk must be updated every time any of the *mn* data disks gets an update. As a result, the technique only applies to archival data and only after they have become immutable.

We propose another solution that spreads the duties of the superparity disk among all column parity disks, thus allowing more updates to be performed in parallel. In addition, our solution can be extended to the case of rectangular RAID arrays that can tolerate all quintuple disk failures without data loss, thus offering alternatives to both Azure LRC and HDFS-XORBAS codes.

A. Bundles of RAID level 5 arrays

Looking back at Fig. 3, we can see that each of the top *m* rows of the rectangular RAID array consists of *n* data disks and a single parity disk. In other words, each of these rows constitutes a RAID level 4 array [10].

We propose to reorganize these *m* rows and transform them into RAID level 5 arrays. As a result, each disk of these m first rows will contain both data blocks and parity blocks with the data blocks occupying $n/(n+1)$ of the disk capacity and the parity blocks the remaining $1/(n+1)$. Fig. 6 illustrates the concept. Note that we now have $n+1$ column parity disks in the bottom row in order to ensure that all $m(n+1)$ disks that hold data are equally protected. As a result, the superparity blocks that were stored the superparity disk are now evenly distributed among the $n+1$ disks of the bottom row, which now contain both parity and superparity data. All fatal triple failures are eliminated because we can now reconstitute the lost parity data.

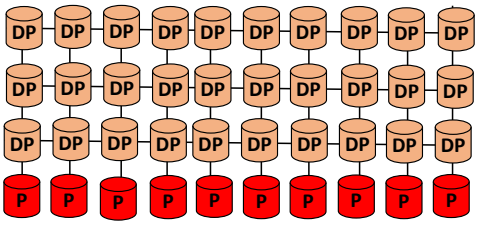

Fig. 8. A bundle consisting of three RAID level 6 arrays with ten disk each and ten additional column parity disks.

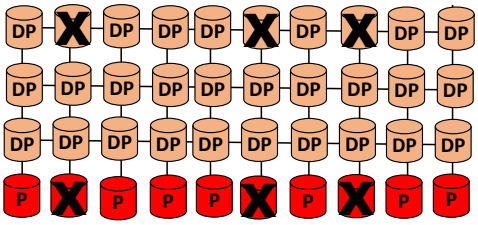

Fig. 9. A fatal sextuple failure.

Parity information will occupy the equivalent of one disk per RAID level 5 array plus $n + 1$ column parity disks. As a result, the parity space overhead of the bundle will be equal to:

$$\frac{m + n + 1}{(m + 1)(n + 1)}.$$

The sole possible fatal quadruple failures involve the failures of four disks that share the same two rows and the same two columns. Fig. 7 shows one of these fatal quadruple failures. As we can see, the four failed disks share the same two rows and the same two columns. Enumerating the set of fatal quadruple failures is thus equivalent to enumerating the number of rectangles that we can form by selecting two arbitrary rows out of $m + 1$ and two arbitrary columns out of $n + 1$, which is equal to $\binom{m+1}{2}\binom{n+1}{2}$. Observing there are $\binom{(m+1)(n+1)}{4}$ possible quadruple failures, the probability that an arbitrary quadruple disk failure will result in a data loss is:

$$\frac{\binom{m+1}{2}\binom{n+1}{2}}{\binom{(m+1)(n+1)}{4}}.$$

Table I displays some selected bundle configurations with their space overheads and the percentage of quadruple disk failures that will result in a data loss. We want to keep the number m of RAID arrays per bundle small in order to involve as few disks as possible in the recovery of single disk failures. At the same time, we want to use longer RAID stripes to keep the parity space overhead under 50 percent.

Comparing the performance of our solution with that of a (6, 2, 2) Azure locally repairable code is difficult because our solution tends to group together many more data disks. The closer we can get is comparing the (6, 2, 2) Azure code with a

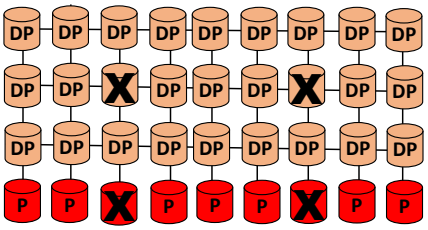

Fig.7. A fatal quadruple failure.

TABLE I. SELECTED RAID LEVEL 5 BUNDLE SIZES AND THEIR RESPECTIVE SPACE OVERHEADS.

Number of RAID stripes	Disks per RAID stripe	Storage capacity (disks)	Space overhead	Fatal quadruple failures
2	8	14	41.7%	0.791%
2	10	18	40.0%	0.493%
2	12	22	38.9%	0.336%
3	8	21	34.4%	0.467%
3	10	27	32.5%	0.295%
3	12	33	31.3%	0.204%
4	8	28	30.0%	0.306%
4	10	36	28.0%	0.195%
4	12	44	26.7%	0.135%
5	8	35	27.1%	0.216%
5	10	45	25.0%	0.138%
5	12	55	23.6%	0.096%

bundle of two RAID level 5 arrays with 4 disks each. Both organizations would comprise six data disks. Our organization would have six parity disks, bringing its space overhead to 50 percent instead of 40 percent for the (6, 2, 2) code. At the same time, our organization would be able to resolve all single disk failures by accessing two disks instead of three and would tolerate 96 percent of quadruple disk failures instead of 86 percent for the (6, 2, 2) code.

Another option would be to bundle together three RAID level 5 arrays with 5 disks each. The organization would have 12 data disks and 8 parity disks, giving it the same space overhead as the (6, 2, 2) code. It would require accessing three disks to resolve all single disk failures and would tolerate 98.8 percent of quadruple disk failures, which is better than the (6, 2, 2) code.

B. Bundles of RAID level 6 arrays

Whenever a higher level of data resiliency is required, we can replace the RAID level 5 arrays used in our scheme by RAID level 6 arrays [2] [14], that is, RAID arrays that tolerate two disk failures.

Fig. 8 represents one such bundle. It consists of three RAID level 6 arrays with ten disks each plus an additional row of ten disks that contain the column parities. Each of the disks in the two RAID level 6 arrays contains both data blocks and parity blocks with data blocks occupying 8/10 of the disk capacity and parity blocks the remaining 2/10. Similarly, the ten disks in the bottom row contain a mixture of parity and superparity blocks.

TABLE II SELECTED RAID LEVEL 6 BUNDLE SIZES AND THEIR RESPECTIVE SPACE OVERHEADS.

Number of RAID stripes	Disks per RAID stripe	Storage capacity (disks)	Space overhead	Fatal sextuple failures
2	10	16	46.7%	0.061%
2	12	20	44.4%	0.034%
2	14	24	42.9%	0.021%
3	10	24	40.0%	0.019%
3	12	30	37.5%	0.011%
3	14	36	35.7%	0.007%
4	10	32	36.0%	0.008%
4	12	40	33.3%	0.004%
4	14	48	31.4%	0.003%
5	10	40	33.3%	0.004%
5	12	50	30.6%	0.002%
5	14	60	28.6%	0.001%

More generally, a RAID level 6 bundle will consist of m RAID level 6 arrays, each containing $n + 2$ disks plus an additional row of $n + 2$ disks containing the column parities of the disks in the *m* RAID level 6 arrays. The space overhead of the bundle will be equal to:

$$\frac{2m + n + 2}{(m + 1)(n + 2)}.$$

As before, we want to keep the number of RAID arrays in the bundle relatively low in order to involve as few disks as possible in the recovery of single disk failure. In the same way, we want to avoid very small RAID level 6 stripes to keep the space overhead below 50 percent.

Since RAID level 6 arrays tolerate all double disk failures, all fatal failures must include three disks in the same RAID array. Not only that, these three disks must be unable to use their column parities to recover their contents. As a result, the bundle will tolerate all quintuple disk failures. As Fig. 9 shows, the sole possible fatal sextuple failures involve the failures of six disks that share the same two rows and the same three columns.

Given there are $\binom{m+1}{2}\binom{n+2}{3}$ possible fatal sextuple failures out of $\binom{(m+1)(n+2)}{6}$ possible sextuple failures, the probability that an arbitrary sextuple disk failure will result in a data loss is equal to:

$$\frac{\binom{m+1}{2}\binom{n+2}{3}}{\binom{(m+1)(n+2)}{6}}$$

Table II displays some selected bundle configurations with their space overheads and the percentage of sextuple disk failures that will result in a data loss. Note that we selected somewhat larger values for the number of disks per RAID level 6 array to compensate for having now the equivalent of two parity disks in each array.

The closest equivalent to an HDFS-XORBAS configuration with ten data disks and six parity disks would be a bundle of two RAID level 6 arrays with seven disks each. The two organizations would have the same storage capacity but our bundle would require five additional parity disks bringing its space overhead to 52 percent. At the same time, it would be able to recover from all single disk failures by accessing two disks, instead of five, and would tolerate all quintuple disk failures and 99.8 percent of sextuple disk failures, instead of all quadruple disk failures and most quintuple failures. As Table II indicates, lowering bundle space overhead requires tripling or quadrupling the capacity of the bundles.

IV. DISCUSSION

A main advantage of our approach is its flexibility. First, we can vary the number *m* of RAID arrays per bundle with smaller values of *m* resulting in both smaller single failure recovery overheads and larger parity space overheads. Second, we can arbitrarily select the sizes of the original RAID arrays. Larger RAID array sizes will both reduce the space overhead and increase the total size of each bundle. In addition, we can select RAID level 5 arrays to obtain triple failure protection or dual-parity RAID level 6 arrays to obtain a bundle that would tolerate quintuple failures.

In the rare situations where even higher levels of data protection are required, we could use STAR arrays [5] to construct bundles that would tolerate up to seven simultaneous disk failures. This could be done without bringing the parity space overhead of the bundle over 50 percent by selecting relatively large STAR arrays. The real drawback of the organization would be its very high update cost, as each block update would have to be propagated to three parity blocks in its STAR array plus four parity blocks in the parity row, resulting in a write amplification factor of eight.

Another advantage of our proposal is its simplicity. Since we are using a two-dimensional design with distinct row and column parities, there are no weights to adjust in the expressions defining the parity calculations. As a result, estimating the reliability of a given bundle configuration is a trivial process.

Finally, we should note that our proposal considers bundling RAID arrays and attaching to them additional parity disks while both Azure LRC and HDFS-XORBAS codes deal with individual blocks. This distinction is not as meaningful as it may appear as the same bundles could be constructed from independent blocks as long as each block resides on a separate disk. (In the same way both Azure LRC and HDFS-XORBAS codes could be used to define fault-tolerant disk array organizations.)

V. CONCLUSION

We have presented a two-dimensional RAID organization aimed at minimizing the number of disks involved in the repair of single disk failures. Our proposal groups together a small number *m* of conventional RAID arrays into a bundle and adds column-wise parity disks such that no two disks in a given RAID array belong to the same parity stripe. As a result, the bundle will be able to recover from any single disk failure by XORing the contents of exactly *m* disks. In addition, we showed that bundles of RAID level 5 arrays could recover without data loss from all triple and at least 96 percent of

quadruple disk failures while bundles of RAID level 6 arrays could similarly recover from all quintuple and at least 99.9 percent of sextuple disk failures.

More work is still needed to evaluate the cost of repairing double and triple failures and estimating the impact of irrecoverable read errors on the repair process.

REFERENCES

[1] B. Beach, "BackBlaze open sources Reed-Solomon erasure coding source code," https://www.backblaze.com/blog/reed-solomon/, June 16, 2015, retrieved June 20, 2018.

[2] W. A. Burkhard and J. Menon, "Disk Array Storage System Reliability," Proc. 23rd Int. Symp. on Fault-Tolerant Computing (FTCS-23), pp. 432–441, June 1993.

[3] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. +, 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," Proc. 23rd ACM Symposium on Operating Systems Principles (SOSP '11), Cascais, Portugal, Oct. 2011.

[4] S. Ghemawat , H. Gobioff ,and S.-T. Leung, "The Google file system," Proc.19th ACM Symp. on Operating Systems Principles (SOSP '03), Bolton Landing, NY, Oct. 2003.

[5] C. Huang and L. Xu, STAR: An efficient coding scheme for correcting triple storage node failures. IEEE Transactions on Computers, Vol. 57, No. 7, pp.889–901, July 2008.

[6] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S,Yekhanin, "Erasure coding in Windows Azure storage," Proc. 2012 USENIX Annual Technical Conf. (USENIX ATC 12), Boston, MA, pp. 15-26, 2012.

[7] 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," Proc 10th USENIX Conf. on File and Storage Technologies (FAST '12), San Jose, CA, Feb. 2012.

[8] T. Miyamae, T. Nakao, and K. Shiozawa, "Erasure code with shingled local parity groups for efficient recovery from multiple disk failures," Proc. 10th USENIX Workshop on Hot Topics in System Dependability (HotDep '14) Broomfield, CO, Oct. 2014.

[9]. J.-F. Pâris, T. Schwarz, S. J., A. Amer and D. D. E. Long, "Highly Reliable Two-Dimensional RAID Arrays for Archival Storage," Proc, 31st Int. Performance of Computers and Communication Conf (IPCCC 2012*)*, Austin, TX, pp. 324–331, Dec. 2012.

[10] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," Proc. 1988 ACM SIGMOD International Conf. on Management of Data, Chicago, IL, pp. 109–116, June 1988.

[11] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, K. Ramchandran, "A Hitchhiker's Guide to Fast and Efficient Data Reconstruction in Erasure-coded Data Centers," Proc. 2014 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), Chicago, IL, Aug. 2014.

[12] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields", Journal of the Society for Industrial and Applied Mathematics (SIAM), 8 (2): 300–304, 1960.

[13] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "XORing elephants: novel erasure codes for big data," Proc. of the VLDB Endowment, Vol. 6, No. 5, pp. 325-336, 2013.

[14] T. Schwarz, S. J., Reliability and Performance of Disk Arrays, PhD Dissertation, Department of Computer Science and Engineering, University of California, San Diego, 1994.

[15] T. Schwarz, S. J., A. Amer, J.-F. Pâris, "Combining low IO-operations during data recovery with low parity overhead in two-failure tolerant archival storage systems," Proc. 21st IEEE Pacific Rim International Symp. on Dependable Computing (PRDC '15), Zhangjiajie, China, Nov. 2015.

[16] A. Wildani, T. J. E. Schwarz, E. L. Miller and D. D. E. Long, "Protecting against rare event failures in archival systems," Proc. 17th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '09), London, GB, pp. 246–256, Sep. 2009.