# Codes for RAID Solutions based upon SSDs

**Alexander Vardy**,* and **Eitan Yaakobi**†
*University of California San Diego, La Jolla, CA 92093, USA
†Computer Science Department, Technion – Israel Institute of Technology, Haifa 32000, Israel
**avardy@ucsd.edu**, **yaakobi@cs.technion.ac.il**

***Abstract*—One of the prominent properties of flash memories is their asymmetry between writing and erasing. When pages, which are the smallest write unit, are updated, they are written in a new copy rather than in place. As a result, every page can have more than one copy in the memory, its current version as well as some of its old invalid copies. Each invalid copy can be cleaned only when the block in which it resides is erased (blocks are the smallest erase unit and are typically in the order of hundreds of pages). This write property introduces redundancy in the memory, given by the invalid copies of the pages, and as a result can also affect the memory lifetime.**

**In this paper we show how this inherent redundancy of invalid pages can be taken advantage of for the purpose of improving RAID solutions which are based upon Solid State Drives (SSDs). Our main contribution in the paper is a construction which shows how to improve the repair bandwidth of codes which are implemented on SSDs. We first show that with a single parity it is possible to transmit on the average roughly half of the data for rebuilding a single drive failure. We then show how these ideas can be extended for Zigzag codes with two parities and again improve their repair bandwidth.**

## I. Introduction

Solid State Drives (SSDs), built upon NAND flash, have become a prominent storage medium. They have several advantages over Hard Disk Drives (HDDs) such as better I/O performance, lowe energy consumption, and better shock resistance. Furthermore, since the price of flash continues to drop down, SSDs have become also an attractive solution for personal computers as well as servers and data centers.

When SSDs are used in servers and data servers, RAID (redundant array of independent disks) [19] is used in order to provide reliability and support drive failures. There is an enormous amount of literature on RAID and over the last three decades several works studied possible solutions which improve different parameters such fault tolerant capability, computation complexity, I/O performance, update complexity, and recently rebuilding complexity.

Existing RAID solutions which were originally designed for HDDs can be used for SSDs as well. However, there are several significant differences between SSDs and HDDs. Mainly, flash suffers a limited life time which depends upon the number of times it can be erased. Hence, several works considered this behavior in order to determine whether to distribute the parities evenly or unevenly across the drives; see e.g. [2], [14], [15]. Other works study optimization of other parameters such as how to efficiently rewrite the data and parities to the drive to reduce the number of block erasures [1], [8], [10]–[13], [16], [17].

The works mentioned above take into account the inherent asymmetry in flash between programming and erasing. The smallest write unit in flash is a *page* and pages are organized into *blocks*, which are the smallest erase unit. In order to avoid redundant block erases upon page rewrites, the pages are rewritten *out-of-place* and a *flash translation layer (FTL)* maps between the physical location of every logical page. Furthermore, in order to accommodate this write mechanism, SSDs have more physical than logical memory and the ratio between the additional memory and logical memory is called *over-provisioning (OP)* [3], [7].

Recently, several works in the area of distributed storage studied codes which their main goal is to optimize the rebuilding process of a single drive failure [4]. Assume that a RAID solution uses a code which can support multiple drive failures, then the goal is to optimize the amount of data which needs to be read and transmitted in order to rebuild the failed drive. Examples of such codes are *regenerating codes*, see e.g. [5], [20], [24]–[26] or *locally repairable codes* [18], [21], [23]. In regenerating codes the main figure of merit, called *repair bandwidth*, is the amount of data that needs to be transmitted in order to rebuild a single drive failure. *Zigzag codes* are one example of such a code. For example, if there are two parity drives and a single drive fails then only half of the data from each of the surviving drives is transmitted in order to rebuild the failed drive.

In this paper we also aim in improving the repair bandwidth for a single drive failure, however we show how to accomplish this goal using the inherent redundancy due to the OP in the SDDs. Let us consider a RAID 5 system, that is a single parity. Traditionally, if one drive fails then the information from the remaining drivers has to be read in order to rebuild this drive. However, when using SSDs, we will show how it is possible to read from significantly less drives in order to rebuild the failed drive.

The rest of this paper is organized as follows. In Section II, we set the model, assumptions, and problems we study in the paper. In Section III we give our first code construction. We show that even with a single parity it is

possible to improve the repair bandwidth such that approximately only half of the data is transmitted for rebuilding. In Section IV, we show how it is possible to accomplish the same task with two redundancy drives. Here we build upon Zigzag codes which achieve the optimal repair bandwidth and show how their repair bandwidth can be improved when using SSDs. Finally, Section V concludes the paper and suggests problems for future research.

## II. Background Definitions and Preliminaries

In this section we formally define the setup, assumption, and problems studied in the paper. For a positive integer $n$ we denote by $[n]$ the set $\{1,\ldots,n\}$.

The smallest write unit in flash memories is a page which its size ranges between 2KB and 16KB [3], [7]. The smallest erase unit is a block which consists of a group of pages, usually in the order of hundreds of pages. Hence, when a page is updated it is not written in its current physical location but rather in a new available location. This *out-of-place* write procedure requires a specific design of the memory. First, there is more physical than logical memory, and the ratio between the amounts of additional storage and logical storage is called *over-provisioning (OP)*. A *Flash Translation Layer (FTL)* maps between the physical location of every logical page. Finally, since pages are always rewritten in a new location, it is required to free space when there is no more available space, in a process called *Garbage Collection (GC)* [3], [7].

Since pages are rewritten out-of-place, a logical page may have more than one copy in the memory, its new updated data as well as some old versions of it. The number of copies of the page depends upon the amount of OP, GC algorithm, and the write workload to the memory. Typical OP ratios for enterprise storage are in the order of 25-30% [22]. Hence, it is not common that *every* logical page has more than one copy in the memory since that will require an OP ratio of at least 100%. However, it is very likely to assume that there exists a time window after the page in rewritten in which its old copy still exists in the memory.

To simplify our notation, we assume that every logical page can have at most two physical copies in the memory, the updated version and its version right before that update. We will later explain when and how we use the previous copies of the pages in the memory. Furthermore, in order to use previous copies of the pages in the memory, we will also need to know how to track their locations. This can be done either by the FTL or by the metadata of the current valid copy of the page.

Let us now describe the RAID structure using multiple SSDs. Assume that there are $n$ SSDs. Every SSD has its own GC algorithm and FTL which we will not change. A linear $[n,k]$ linear code will be called a *RAID code* if there are $k$ information drives, $r = n-k$ redundancy drives, and it can tolerate $r$ drive failures. A codeword will be called also a *stripe* and it will consist of one page from each drive. We will denote a stripe by

$$(a_1,\ldots,a_k,p_1,\ldots,p_r),$$

where $a_i$ for $i \in [k]$ denotes an *information page* and for $j \in [r]$, $p_j$ is a *redundancy page*. We assume also that on each write at most a *single* page in a stripe is updated. We will also assume that the redundancy pages are distributed evenly among the drives.

The *repair bandwidth* is defined to be the amount of data that needs to be transmitted in order to rebuild a failed drive [4], [26], and the *repair bandwidth ratio* is the ratio between the amount of transmitted data and the available data (i.e. the data that didn't fail). While in most works the repair bandwidth and ratio are studied in the *worst case* we study here the *average case* in which we will calculate the average amount of data which is transmitted to rebuild a single drive failure, where every drive fails with the same probability.

## III. Average Repair Bandwith with a Single Parity

In this section we study $[n,n-1]$ RAID codes, so each stripe has $n-1$ information pages and one more redundancy page. Under the conventional setup, if one drive fails then for each stripe, all the remaining pages have to be transmitted to rebuild the failed page. In this case we will say that the repair bandwidth is $n-1$ pages and the repair bandwidth ratio is 1. The main goal of this section is to show that when using SSDs, it is possible to have better average repair bandwidth.

We first describe the code construction with a single redundancy page. To simplify some of the analysis here, we assume that $n$ is odd in the rest of this section. A crucial part of the the construction is the update policy in which we rewrite the redundancy page upon a rewrite of an information page. Let us denote the $n$ pages in a stripe as a codeword

$$(a_1,a_2,\ldots,a_{n-1},p),$$

so for $i \in [n-1]$, $a_i$ is an information page and $p$ is a redundancy page. The redundancy page will be calculated according to two possible options. It can be either a simple parity page, in which case we will say that it is a redundancy page of *type I*,

$$p = \sum_{i=1}^{n-1} a_i,$$

or, it can be calculated according

$$p' = \sum_{i=1}^{(n-1)/2} a_i + \alpha \cdot \left( \sum_{i=(n+1)/2}^{n-1} a_i \right),$$

where $\alpha$ is a primitive element in $GF(4)$, and then we will

say that the redundancy page is of *type II*[1].

At the beginning, the redundancy page is calculated to be of type I, on the next page update the redundancy page is calculated to be of type II, on the next update of type I and so on. This code construction with the update policy will be called an $[n, n-1]$ *SSD-RAID code*. In the next theorem we will analyze the repair bandwidth of this code.

**Theorem 1.** *Assume that the failed page in the $[n, n-1]$ SSD-RAID code is neither the redundancy page nor the last updated information page, then the repair bandwidth to recover the failed page is at most $(n-1)/2+3$ pages.*

*Proof:* Assume the codeword written in the stripe is

$$(a_1, a_2, \ldots, a_{n-1}, p),$$

and the first page was rewritten so the codeword is updated to be

$$(a'_1, a_2, \ldots, a_{n-1}, p').$$

Assume also without loss of generality that $p$ is of type I and $p'$ is of type II. We have the following two equations:

$$p = \sum_{i=1}^{n-1} a_i,$$
$$p' = a'_1 + \sum_{i=2}^{(n-1)/2} a_i + \alpha \cdot \left( \sum_{i=(n+1)/2}^{n-1} a_i \right).$$

Let us denote $A = \sum_{i=2}^{(n-1)/2} a_i$ and $B = \sum_{i=(n+1)/2}^{n-1} a_i$. Thus, we get the following two equations

$$p + a_1 = A + B,$$
$$p' + a'_1 = A + \alpha B.$$

Since $\alpha$ is a primitive element in $GF(4)$ these two equations are linearly independent and we can recover the value of both $A$ and $B$, while reading 4 pages so far $a_1, a'_1, p, p'$. The variable $A$ is a summation of $(n-1)/2-1$ pages and $B$ is a summation of $(n-1)/2$ pages. Let $i$ be the index of the failed drive. If $2 \leqslant i \leqslant (n-1)/2$, then it is enough to read $(n-1)/2-2$ pages to recover the failed page and if $(n+1)/2 \leqslant i \leqslant (n-1)$ then it is enough to read $(n-1)/2-1$ pages to recover the failed page. In either case, we read at most

$$4 + (n-1)/2 - 1 = (n-1)/2 + 3$$

pages to rebuild the failed page. ∎

We can deduce the following corollary on the average repair bandwidth ratio.

[1]For that we will perform the operations on every pair of bits together from each page.

**Corollary 2.** *The average repair bandwidth ratio of the $[n, n-1]$ SSD-RAID code is*

$$\frac{1}{2} + \frac{7n-11}{2n(n-1)}.$$

*Proof:* If the failed page is the last updated page or the parity page then $n-1$ pages are read for rebuilding and the repair bandwidth ratio is 1. Otherwise, we saw from Theorem 1 that for $(n-1)/2-1$ pages the repair bandwidth is $(n-1)/2+2$ pages and for the remaining $(n-1)/2$ pages the repair bandwidth is $(n-1)/2+3$ pages. Thus, we conclude that the average repair bandwidth ratio is

$$\begin{aligned} &\frac{2}{n} \cdot 1 + \frac{(n-1)/2-1}{n} \cdot \frac{(n-1)/2+2}{n-1} \\ &+ \frac{(n-1)/2}{n} \cdot \frac{(n-1)/2+3}{n-1} \\ &= \frac{n^2+6n-11}{2n(n-1)} = \frac{1}{2} + \frac{7n-11}{2n(n-1)}. \end{aligned}$$

∎

Note that the number of stripes is very large and we also assume that the redundancy pages are distributed evenly among the drives. Thus, if pages are also rewritten evenly among the drives then the worst case repair bandwidth ratio for each SSD will be very close to the average value we calculated here. We also note that in case the previous copy of the redundancy page or the last updated information page in the stripe were cleaned by the GC, then we can simply invoke the conventional recovery procedure with repair bandwidth ratio 1. Furthermore, even with this modification we can always support a single drive failure. The same claims will hold also in the next section where we give a similar construction with two redundancy pages.

## IV. Average Repair Bandwith with Two Parities

The idea we presented in Section III can be extended for multiple drive failures. We will consider here codes which tolerate two drive failures while the extension for multiple drive failures will be left for future work.

In general, if one uses Reed Solomon (RS) codes as an $[n, n-2]$ RAID code, then the repair bandwidth for a single drive failure is $n-2$ pages, so the rebuilding bandwidth ratio is $(n-2)/(n-1)$. We could improve the average ratio so it will be roughly the same value we derived in Corollary 2. However we seek to have a better result.

Our point of departure in this section is the recent construction of *Zigzag codes* by Tamo et al. [24]. Zigzag codes are a family of codes which optimize the repair bandwidth. In particular, they can be used to construct $[n, n-2]$ array codes with optimal repair bandwidth ratio $1/2$. Specifically, if one drive fails then it is enough to read and transmit half of the remaining data from each drive in order to rebuild the failed drive. We will show how, using SSDs, it is possible

to improve this repair bandwidth ratio. Let us first explain the main ideas of the construction of Zigzag codes.

Zigzag codes are another family of array codes. If there are two parity drives, then every entry in the array has to belong to a field of size at least 3. Thus, we will assume that every entry is a pair of bits and thus the column can be considered to be a page. One of the constraints of Zigzag codes is that the number of rows in the array has to be exponential in the number of drives. If a page is of size 16KB then, this allows having $2^{16}$ rows and thus 16 drives. Clearly, this number can be improved if the size of each page is greater than 16KB or if we combine several pages together. For the purpose of the analysis, we will assume that the number of drives is moderate and thus is not a concern to the construction of Zigzag codes. Furthermore, we note that there are other similar constructions to Zigzag codes with optimal repair bandwidth such as [5], or other codes with repair bandwidth ratio less than 1, such as EVENODD codes with repair bandwidth ratio 3/4 [25], that can also be used in our constructions. We choose here Zigzag codes so illustrate our ideas.

Since we treat every column as a page, we can consider the Zigzag code as an $[n, n-2]$ RAID code which its repair bandwidth ratio is $1/2$. We denote the codewords in this code as a stripe

$$(a_1, \ldots, a_{n-2}, p_1, p_2).$$

The redundancy page $p_1$ is a simple parity of the information pages so $p_1 = \sum_{i=1}^{n-2} a_i$. To understand the rule in which the second redundancy page is updated, denote $p_2 = (p_{2,1}, \ldots, p_{2,m})$, where $m$ is the number of pair of bits in $p_2$, and we write similarly for $i \in [n-2]$ $a_i = (a_{i,1}, \ldots, a_{i,m})$. Then the construction of Zigzag codes determines for each $j \in [m]$ a linear function

$$p_{2,j} = \sum_{i=1}^{n-2} \gamma_{i,j} a_{i,\sigma(i,j)}. \tag{1}$$

where the values of $\gamma_{i,j}$ and $\sigma(i,j) \in [m]$ are set by the code construction and the coefficients $\gamma_{i,j}$ need to belong to a field of size at least 3 (in our case we simply use the field $GF(4)$).

We will follow the same ideas as in the construction of the $[n, n-1]$ SSD-RAID code from Section III to update the redundancy pages. For simplicity, in this section we will assume that $n$ is even. The redundancy pages $p_1, p_2$ are updated in the same way to be either of type I or type II. For the redundancy page $p_1$ we use exactly the same rule as in the $[n, n-1]$ SSD-RAID code. For the second redundancy page, if it is calculated to be of type I then it is calculated according to equation (1) and if it is of type II then it is calculated according to the rule

$$p_{2,j} = \sum_{i=1}^{n/2-1} \gamma_{i,j} a_{i,\sigma(i,j)} + \alpha \cdot \left( \sum_{i=n/2}^{n-2} \gamma_{i,j} a_{i,\sigma(i,j)} \right),$$

where $\alpha$ is again a primitive element in $GF(4)$.

We call this code construction an $[n, n-2]$ SSD-RAID code, and in the next theorem we will analyze its repair bandwidth.

**Theorem 3.** *Assume that the failed page in the $[n, n-2]$ SSD-RAID code is neither the redundancy page nor the last updated page, then the total amount of transmitted data to recover the failed page equals to at most $n/4 + 3$ pages.*

*Proof:* Assume the following stripe is a codeword written in the drives

$$(a_1, a_2, \ldots, a_{n-2}, p_1, p_2),$$

where $p_1$ and $p_2$ are of type I. Assume without loss of generality that the first page was rewritten so the codeword is updated to be

$$(a'_1, a_2, \ldots, a_{n-2}, p'_1, p'_2),$$

and $p'_1$ and $p'_2$ are of type II. As before we denote $A = \sum_{i=2}^{n/2-1} a_i$ and $B = \sum_{i=n/2}^{n-2} a_i$, so we can recover both $A$ and $B$ by reading 4 pages, namely $a_1, a'_1, p_1, p'_1$. According to the Zigzag construction we can write for $j \in [m]$, $p_{2,j} = \sum_{j=1}^{n-2} \gamma_{i,j} a_{j,\sigma(i,j)}$ and for $p'_2$ we similarly have for $j \in [m]$

$$p'_{2,j} = \gamma_{1,j} a'_{1,\sigma(i,j)} + \sum_{i=2}^{n/2-1} \gamma_{i,j} a_{i,\sigma(i,j)} + \alpha \cdot \left( \sum_{i=n/2}^{n-2} \gamma_{i,j} a_{i,\sigma(i,j)} \right).$$

Next, for $j \in [m]$ we denote $c_j = \sum_{i=2}^{n/2-1} \gamma_{i,j} a_{j,\sigma(i,j)}$ and $d_j = \sum_{i=n/2}^{n-2} \gamma_{i,j} a_{j,\sigma(i,j)}$ then again as before we can recover the values of $c_j$ and $d_j$ since we get two linearly independent equations. Finally we construct the vectors $C = (c_1, \ldots, c_m)$ and $D = (d_1, \ldots, d_m)$.

Let $i$ be the index of the failed page. If $2 \leqslant i \leqslant (n-2)/2$, then we consider the following codeword in the $[n, n-2]$ Zigzag code

$$(0, a_2, \ldots, a_{n/2-1}, 0, \ldots, 0, p^*_1, p^*_2).$$

We can verify that for this codeword $p^*_1 = A$ and $p^*_2 = C$. Hence, we can apply the decoding procedure of the Zigzag code in order to rebuild the page $a_i$, while from each drive we need to transmit only half of the page. However, we need to do so only for $n/2 - 3$ drives (the values of $A$ and $C$ are already known). Therefore, the total amount of transmitted data in pages is

$$4 + (n/2 - 3)/2 = n/4 + 5/2.$$

Similarly, we can derive that for $n/2 \leqslant i \leqslant n-2$ the total amount of transmitted data in pages is

$$4 + (n/2 - 2)/2 = n/4 + 3.$$

■

As a consequence of the last theorem we conclude the following corollary on the average repair bandwidth.

**Corollary 4.** *The average repair bandwidth ratio of the* $[n, n-2]$ *SSD-RAID code is*

$$\frac{1}{4} + \frac{15n - 38}{4n(n-1)}.$$

*Proof:* If the failed page is the last updated page or one of the two redundancy pages then half of each of the remaining $n-1$ pages is transmitted for rebuilding and the repair bandwidth ratio is 1/2 in this case. Otherwise, from Theorem 3, for $n/2-2$ pages the repair bandwidth ratio is $\frac{n/4+5/2}{n-1}$ and for $n/2-1$ pages the repair bandwidth ratio is $\frac{n/4+3}{n-1}$. Together we get that the average repair bandwidth ratio is

$$\begin{aligned}&\frac{3}{n} \cdot \frac{1}{2} + \frac{n/2-2}{n} \cdot \frac{n/4+5/2}{n-1} + \frac{n/2-1}{n} \cdot \frac{n/4+3}{n-1} \\ =&\frac{1}{4} + \frac{15n-38}{4n(n-1)}.\end{aligned}$$

■

We finally note that the modifications we introduced in the construction of Zigzag do not affect their erasure correction capability of two drive failures.

## V. Conclusion

The main goal of this work is to show how the inherent redundancy in SSDs due to over-provisioning can be used in order to improve the performance of RAID systems. The main feature which we focused on here is the repair bandwidth, where we showed that even with a single parity it is possible to significantly improve the repair bandwidth by a factor of almost 2. We believe that the examples and constructions we gave in the paper are only first ideas for more codes which use similar ideas and properties from flash memories to improve the performance of codes for distributed storage. In particular, these ideas can be used in other regenerating codes as well as other families of codes such as locally repairable codes [6], [23] in order to further improve their locality.

## References


[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," *Proc. of USENIX ATC*, Jun 2008.

[2] M. Balakrishnan, A. Kadav, V. Prabhakaran, and D. Malkhi, "Differential RAID: Rethinking RAID for SSD Reliability," *ACM Trans. on Storage*, vol. 6, pp. 4, Jul. 2010.

[3] P. Desnoyers, "What systems researchers need to know about NAND flash," *5th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, 2013.

[4] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

[5] E. En Gad, R. Mateescu, F. Blagojevic, C. Guyot, and Z. Bandic, "Repair-optimal MDS array codes over GF(2)," *Proc. IEEE Int. Symp. on Inf. Theory*, pp. 887–891, Istanbul, Turkey, Jul. 2013.

[6] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Trans. Inf. Theory*, vol. 58, no. 11, pp. 6925–6934, Nov. 2012.

[7] L.M. Grupp, A.M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P.H. Siegel, and J.K.Wolf, "Characterizing flash memory: anomalies, observations, and applications," *Proc. 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 24–3, New York, New York, Dec. 2009.

[8] S. Im and D. Shin, "Flash-aware RAID techniques for dependable and high-performance flash memory SSD," *IEEE Trans. on Computers*, vol. 60, pp. 80–92, Jan 2011.

[9] N. Jeremic, G. Muhl, A. Busse, and J. Richling, "The pitfalls of deploying solid-state drive RAIDs," *SYSTOR*, 2011.

[10] J. Kim, J. Lee, J. Choi, D. Lee, and S. H. Noh, "Enhancing SSD reliability through efficient RAID support," *Proc. of APSys*, Jul 2012.

[11] J. Kim, J. Lee, J. Choi, D. Lee, and S. H. Noh, "Improving SSD reliability with RAID via elastic striping and anywhere parity," *Proc. 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 1–12, Budapest, Jun. 2013

[12] S. Lee, B. Lee, K. Koh, and H. Bahn, "A Lifespan-aware reliability scheme for RAID-based flash storage," *Proc. of ACM Symp. on Applied Computing*, SAC 11, 2011.

[13] Y. Lee, S. Jung, and Y. H. Song, "FRA: a flash-aware redundancy array of flash storage devices," *Proc. of ACM CODES+ISSS*, Oct 2009.

[14] Y. Li, P. P. C. Lee, and J. C. S. Lui, "Stochastic analysis on RAID reliability for solid-state drives," Technical Report. http://arxiv.org/abs/1304.1863.

[15] Y. Li, P. P. C. Lee, and J. C. S. Lui, "Stochastic modeling of large-scale solid-state storage systems: analysis, design tradeoffs and optimization," *Proc. of SIGMETRICS*, 2013.

[16] B. Mao, H. Jiang, S. Wu, L. Tian, D. Feng, J. Chen, and L. Zeng, "HPDA: A hybrid parity-based disk array for enhanced performance and reliability," *ACM Trans. on Storage*, vol. 8, pp. 4, Feb. 2012.

[17] K. Park, D.-H. Lee, Y. Woo, G. Lee, J.-H. Lee, and D.-H. Kim, "Reliability and performance enhancement technique for SSD array storage system using RAID mechanism," *IEEE Int. Symp. on Comm. and Inform. Tech.*, 2009.

[18] L. Pamies-Juarez, H.D.L. Hollmann, and F. Oggier, "Locally repairable codes with multiple repair alternatives," *Proc. IEEE Int. Symp. on Inf. Theory*, pp. 892–896, Istanbul, Turkey, Jul. 2013.

[19] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," *SIGMOD*, Jun. 1988.

[20] K. Rashmi, N. Shah, and P. Kumar, "Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5227–5239, Aug. 2011.

[21] A. S. Rawat, D.S. Papailiopoulos, A.G., Dimakis, and S. Vishwanath, "Locality and availability in distributed storage" arXiv:1402.2011v1, Feb. 2014.

[22] K. Smith, "Understanding SSD over-provisioning," *EDN Network*, January 2013.

[23] I. Tamo and A. Barg, "A family of pptimal locally recoverable codes," *IEEE Trans. on Inf. Theory*, vol. 60, no. 8, pp. 4661–4676, Aug. 2014.

[24] I. Tamo, Z. Wang, and J. Bruck, "Zigzag codes: MDS array codes with optimal rebuilding," *IEEE Trans. on Inf. Theory*, vol. 59, no. 3, pp. 1597–1616, Mar. 2013.

[25] Z. Wang, A. G. Dimakis, and J. Bruck, "Rebuilding for array codes in distributed storage systems," *Proc. IEEE GLOBECOM Workshops*, Dec. 2010, pp. 19051909.

[26] Y. Wu, A. Dimakis, and K. Ramchandran, "Deterministic regenerating codes for distributed storage," presented at the *45th Allerton Conf. Control, Comput., Commun.*, Monticello, IL, 2007.