Combining Low IO-Operations During Data Recovery with Low Parity Overhead in Two-Failure Tolerant Archival Storage Systems

Thomas Schwarz, S.J.
Universidad Centroamericana
La Libertad, El Salvador
tschwarz@uca.edu.sv

Ahmed Amer
Santa Clara University
Santa Clara, CA, USA
aamer@scu.edu

Jehan-François Pâris
University of Houston
Houston, TX, USA
jfparis@uh.edu

***Abstract*—Archival data storage systems contain data that must be preserved over long periods of time but which are often unlikely to be accessed during their lifetime. The best strategy for such systems is to keep their disks powered-off unless they have to be powered up to access their contents, to reconstruct lost data, or to perform other disk maintenance tasks. Of all such tasks, reconstructing data after a disk failure is the one that is likely to have the highest energy footprint and the most impact on the overall power consumption of the array, because it typically involves powering up all the disks belonging to the same reliability stripe as the failed disk and keeping them running for considerable time at each occurrence.**

We investigate two two-failure tolerant disk layouts that have lower parity overhead than the number of disks read (and hence powered-on) for recovering data on lost drives would suggest. Our first organization is a flat XOR code that organizes the data disks into a rectangle with fewer rows than columns, and adds a simple parity disk to each row and column. Recovery from a disk failure proceeds by prefering columns when reconstructing lost data, and thereby has fewer reads than the parity overhead would normally suggest.

Our second layout is based on the most basic pyramid code. We can view this layout as an example RAID Level 6 variant. In this variant, a stripe has a Q-parity calculated from the data disks in the stripe, but the data disks are also organized into smaller groups where each group has a separate P-parity calculated as the exclusive-or of the data disks in the group.

We compare the two layouts by measuring their robustness to data loss, their one-year survival rate, and the expected number of number of disks that must be involved to recover from both single and multiple disk failures. Our results show that rectangular layouts are significantly more reliable than layouts based on the most basic Pyramid codes, but that they also require more disk accesses to recover from disk failures.

I. Introduction

Flash technology has conquered an important segment of the storage market and new storage technologies such as phase-change memories hold great promise for the future. In the near and middle term, though, the attractive price-capacity ratio of magnetic storage technology will mean that a large portion of the enormous amount of data generated (2.8 zettabytes in 2012) will use it for storage, especially for archival data.

Much of this data has an unknown value and while it will usually never be accessed, its loss can lead to high costs. A typical example is an archive of business records or SCADA data that needs to be presented for litigation.

In an archival storage system, disks would be powered on only for checking their availability, and possibly for data scrubbing. For data recovery, other disks need to be powered on, which increases the energy costs. We investigate here systems for archival storage that provide two-failure tolerance, that have a low ratio $1/t$ of parity to data disks, but which on average would use less than t disks to recover from a disk failure or read error.

We compare two different organizations with respect to their behavior with regards to failed disks. Recovering data on a lost disk implies reading from other disks in the same reliability stripe and thus having to power these disks on for the long period of time it takes to read all their data. On the other hand, we want to limit the amount of parity data that needs to be generated and stored. We are thus faced with two horns of a dilemma. We can make reliability stripes short in order to limit the energy costs of recovery operations or we can make reliability stripes large in order to achieve good ratios of parity over user data. Since latent sector errors are often only encountered when trying to reconstruct data, their presence has been recognized as an important source of data loss. Disk arrays deal with their potential presence by providing two-failure tolerance. Loosely speaking, each data disks is then protected by (at least) two parity disks. This allows us to escape our dilemma. Figuratively speaking, one parity can be calculated from many data disks in order to give good parity overhead while the other can be made to cover a few data disks to make recovery from a single failure very efficient. Our two disk array organizations differ in whether they use parity generation based on exclusive-or operations only or whether they use algebraically more involved codes such as Reed-Solomon codes.

Our first layout uses a flat XOR code and tolerates all double, and most triple, disk failures. The layout organizes $r \times s$ disks into a rectangular array consisting of r rows of s disks each. Each row will have its own row parity disk and each column will have its own column parity disk, for a total of $r+s$ parity disks. The parity overhead is therefore $(r+s)/rs$.

In the case of a single disk failure, this disk can always by repaired using a reliability stripe of length r with additional parity disk. Thus, no more than r disks have to be powered on and read in order to retrieve the data on the failed disk.

The second construction uses the most basic variant of a Pyramid code defined by Huang, Chen, and Li [10]. It can be most easily described as a variant of a RAID Level 6. Each stripe in the RAID Level 6 consists of tu data disks that are grouped into t groups of u data disks each. The Q-parity remains as it is and still applies to the entirety of the stripe. The P-parity disk of the RAID Level 6 stripe is replaced by t P-parity disks, one for each group, that hold the exclusive-or parity of the data disks in this group. For $t = 1$ this is a RAID Level 6 layout with fixed parity disks. All tu data disks in the stripe belong to a Q-parity stripe containing all data disks, but also to a smaller parity stripe with u data disks and a P-parity disk for the group. The stripe thus counts $t+1$ parity disks and its parity overhead is $(1+t)/ut$. Like the preceding layout, our second layout protects its contents against all double and most triple failures. This layout also provides energy-efficient repairs of single disk failures, as these repairs will only require reading data from no more than u disks.

In an archival storage systems, our two layouts have no operational disadvantages over the two-dimensional or the RAID Level 6 layout, respectively. Various studies report that disk drives in a large population fail at an annual rate between 1% to 6% [5], [6], [13], [15], [16], [19]. Higher annual failure rates of more than 60% for a specific batch have been observed in a large installation [2]. Reading a 6TB disk at 150MB/sec takes a little bit more than 11 hours. An installation with 10000 disks has about 500 disk failures to deal with and needs to power up $500r$ disks for at least 11 hours, amounting to $231r$ disk days, where r is the average number of disks read for a data recovery. Using an organization that lowers this average number has a visible impact on the energy consumption of the installation. We have left dealing with individual sector errors out of this calculation because recovery of the data in an unreadable sector only needs to access one sector in, on average, r disks.

Our results should also be of interest in other situation where use of Pyramid codes is appropriate, for example in cloud storage where we might hope to minimize or shape network traffic.

II. Construction

We introduce two layouts for archival disk arrays that provide two-failure tolerance and reduce the energy footprint of disk repairs. To protect against complete or partial failure, we arrange the disks in reliability stripes to which we add one or more parity disks. Using the parity disks and reading other disks in the stripe, we can then recover the data on a completely or partially failed disk and store it elsewhere.

It is important to point out the usefulness of, and justify our focus upon, two-failure tolerance schemes for archival storage systems. It is quite common for a disk to have some sector errors. The landmark study by Baravasundaram *et al* found latent errors in 3.45% of the 1.53 million consumer- and enterprise-class disks studied over a period of 32 months [1]. In general, the number of affected sectors is small and constitutes only a tiny proportion of all sectors on a disk. Therefore, even if all disks in a reliability stripe have sector errors, it is highly unlikely that two or more have the same relative sector corrupted. However, latent sector errors in conjunction with a disk loss in the same reliability stripe make it impossible to recover the data in the corrupted sector or in the failed disk using this stripe. Unfortunately, it is often only the attempt of reconstructing data that reveals a latent sector error on a disk used for the reconstruction. While disk scrubbing [17] can be used to discover latent sector failures early, a frequent scrubbing schedule has high energy cost. The consensus in the storage community appears to be therefore that two-failure tolerance is the minimum prudent tolerance level for large storage systems. Arguments for higher failure tolerance levels can be made, but their persuasiveness depends on the usage and the value of the data. Two-failure tolerance in a storage system for archival data seems to fulfill the due care requirement for data that *might* be requested in litigation.

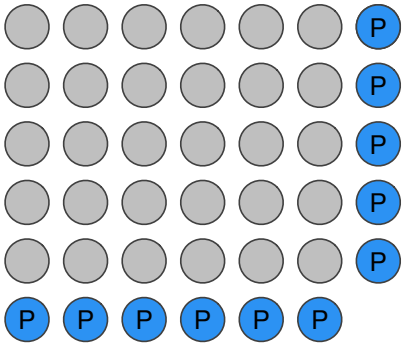

Fig. 1: Rectangular layout with $r = 5$ rows and $s = 6$ columns

A. Rectangular Disk Array Layouts

Two-dimensional RAID arrays as defined by Hellerstein *et al* [8] belong to the vast class of flat XOR codes defined by Greenan *et al* to be codes where parity symbols are calculated from certain subsets of data symbols [7]. We call these subsets reliability stripes.

A two-dimensional RAID array arranges all data disks into r rows and s columns with a data disk in each intersection of a row and a column for a total of rs data disks. Each row and each column contains a parity disk containing the parity of the row or column, respectively (Figure 1). The parity overhead is therefore $(r+s)/rs$. We refer to these layouts as rectangular layouts.

B. RAID Level 6 variant based on a basic Pyramid code

We consider a variant of a RAID Level 6 stripe with P- and Q-parity, presented in Figure 2 that is the simplest instance of a Pyramid code [9] [10]. We calculate the P or exclusive-or parity not of all disks in the stripe, but organize the data disks into m subgroups with n data disks each. We then attach a P-parity disk to each of the m subgroups. This increases the

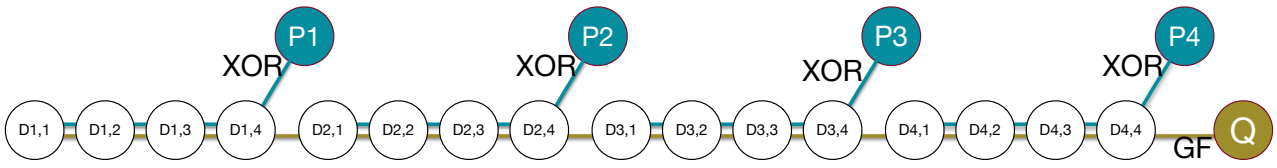

Fig. 2: Representation of the disk array organized by a basic Pyramid code. The organization consists of a single stripe. The data disks are organized in subgroups of size 4. Each subgroup contains a P-parity (with the exclusive-or parity of the disks in the stripe) and the stripe has an additional Q-disk containing the Q-parity calculated from the contents of all data disks in the stripe.

number of parity disks to $m+1$. If a data disk has been lost, we can recover from the local P parity if no other disk in the subgroup has been lost. If a subgroup has two disk failures, then we can recover by using the Q parity. This is only possible if all other failures can be reconstructed locally. Figure 3 shows an extreme case of a failure pattern from which we can recover. In this example, we recover by reading the Q-parity, $n = 5$ disks in the three subgroups with only one failure, and the remaining disks in the subgroup with the double failure. These are mn reads in total. We then calculate the contents of all failed P-parity and data disks in the groups with single failures.

III. Recovery Operations

Data recovery operations can discover sector failures, which would necessitate a change in the recovery procedure or might constitute dataloss. If data recovery is possible at all, we need to read from additional disks. In an archival system, this implies powering on additional disks. However, this is a short operation with little impact on the overall power budget. We therefore focus on the recovery from disk failures. Our two constructions differ with regards to the organization of recovery of data lost on failed drives.

A. Recovery in the rectangular layout

Recovery in the rectangular layout is computationally less involved since we never have to perform Galois field arithmetic, but recovery plans are more involved. In contrast to the RAID Level 6 layout based on the basic Pyramid code, there is often more than one way to recover lost data. This very complexity is an explanation for the superior robustness of this layout as we will see below.

Recovering data from multiple failed disks might lead to cascading recovery operations. In the example given on the left in Figure 4, we have a total of eight failed disks. Despite the failure of more than 12% of all disks, we can still recover all the data. We can clearly reconstruct the parity data on the parity disk in the third column by reading all the data disks in the column. Of the remaining failed disks, only the data on Data Disk 1 can be directly recovered by reading all the disks in the second row. Once this disk is recovered, we can reconstruct the data on Disk 2 reading the other disks in the second column, then the data in Disk 3 by reading the fourth row. This cascading recovery plan then recovers the data on Disks 4, 5, and 6, and finally the data on the parity disk.

Obviously, waiting for all data in Disk 1 to be recovered before starting to recover the data previously in Disk 2, *etc.* means that the execution of the whole recovery plan takes a long time, namely the time to read seven full disks. Also, the amount of data read would not fit in RAM. The obvious solution divides the disk into sections. In this mode, we recover the data on the first section of Disk 1 first. This operation loads the contents of the other disks in the second row into memory. After reconstructing the data and writing it to the replacement disk, we can retain the data sections of disks that are to be read for the recovery of Disks 4 and 6 as well of the column parity disk. When we have reconstructed the data in the first section of Disk 1, we can then start to recover the data in the first section of Disk 2, while starting to work on the second sector of Disk 1. Recovery operations thus proceed in parallel and necessary cascades only slightly retard recovery from all current disk losses.

We use the example on the right of Figure 4 in order to illustrate our greedy algorithm to find a recovery plan. The basic idea is to recover data as much as possible using small stripes. In this example, we can only recover (the contents of) Disk 1 by powering on five disks only. After this recovery, we need to use a stripe of size 10 in order to recover. We can choose to recover either Disk 5, Disk 6 or Disk 7. All three choices result in powering on nine more disks, since we already have the contents of disks in the column of Disk 1 already available. We randomly choose on of the disks, let it be Disk 6. This is a dangerous choice since we will have to recover Disks 7 and 8 eventually and then can recover Disk 6 with a small stripe. After recovery of (the contents of) Disk 6, we can recover Disk 5 or Disk 7. Again, we randomly choose to recover Disk 5. At this point, we can recover Disk 4 using a small stripe. This leaves Disks 7 and Disk 3 that can be recovered, both with ten reads. Let's pick Disk 3. Now, Disk 2 can be recovered powering on five disks. Now we can pick Disk 7 or Disk 8. After choosing one and recover with 10 disks, we can recover the other by powering on five disks. Thus, our schedule is 1 (small), 6 (large), 5 (large), 4 (small), 3 (large), 2 (small), 7 (large), and 8 (small). Because we can reuse already reconstructed or read disk contents, the number of disks powered on is smaller, Figure 5 (left), namely 40.

An alternative recovery program recovers Disk 6 as the last disk. This gives a schedule of recovering the failed disks in order 1, 7, 5, 4, 3, 2, 8 and 6. We read exactly the same number of disks as before, namely 40. We reconstruct the contents of 8 disks and we leave 11 disks powered off. We marked the read disks on the right of Figure 5, and as we can see, the sets

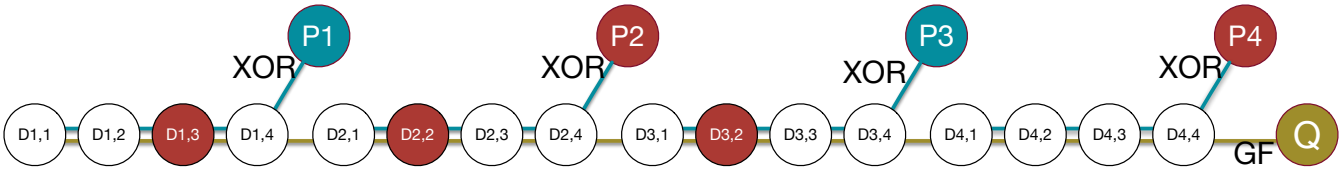

Fig. 3: Reliability stripe with a recoverable pattern of disk failures. Failed disks are shown in red. The *P*- and *Q*-parity disks are marked with a letter P or Q, respectively.

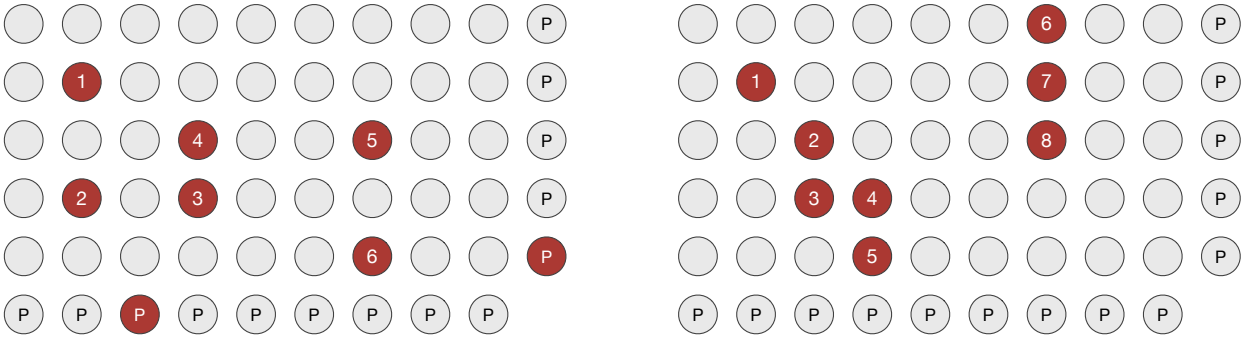

Fig. 4: Recovery examples for the rectangular layout. Red disks are failed data disks.

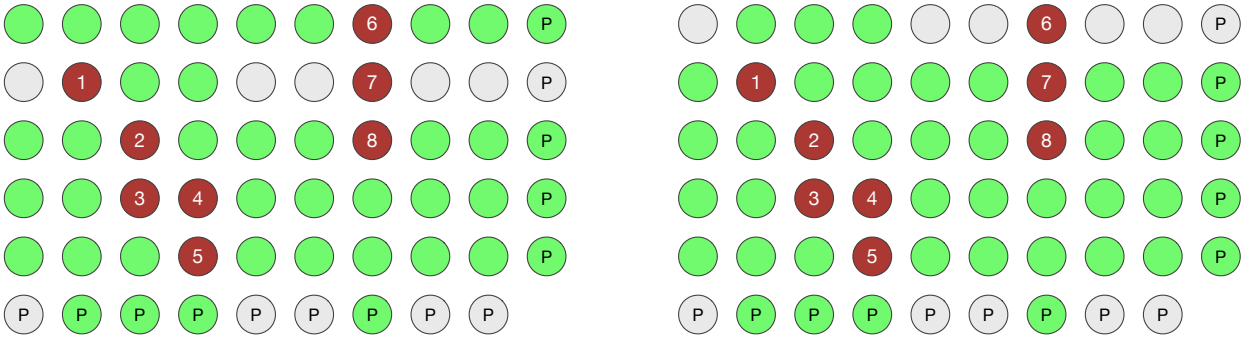

Fig. 5: Read disks in two different recovery schedules in the case depicted in Figure 4, right.

of disks read are different.

For the simulations, we used a greedy strategy that at each point asks which is the disk to be recovered requiring the least number of disks to be powered on. As our example shows, this is not always optimal. A more involved algorithm should be chosen in an industrial implementation, but does not appear to yield major savings.

B. Recovery in the RAID Level 6 layout based on the basic Pyramid code

The Raid Level 6 / Pyramid layout consists of individual stripes and the recovery of data located in one stripe is independent of data located in another stripe. Furthermore, if there is one failure in a subgroup, then we recover by reading from the other disks in the subgroup, independently from operations in other groups. If there is a failure of the *Q*-parity then we need to read n disks in each group in order to reconstruct it. We can use the data gathered to also reconstruct data lost in an individual group. If there is a double failure in a group, we also need to read n disks in each group in order to recover from the double failure, but can also use the data to recover any (single) failure in one or more of the other groups.

Figure 3 gives an example where we have to recover from 4×4 disks since Group 2 contains a double failure. In this example, the three remaining groups have one failure each, but the *Q*-parity survived. We therefore read all remaining disks in the stripe.

IV. Declustered Arrays

Declustering creates equally sized disklets from the disks in a disk array, chooses N disklets from different disks, and arranges these disklets according to a chosen layout with a total of N data and parity disks. It continues to do so until there are no disklets left. The purpose of declustering is to distribute the load of a degraded read (from an unavailable disk) or the load of reconstructing the data on one or more failed disks over all the disks in the array. This allows faster recovery from disk failures and closes the window of vulnerability opened by a disk failure and constituted by the possibility of data loss because of further disk failures before the original disk is repaired [21].

The performance advantage of faster reads in the degraded mode is by definition not realized in archival storage. Indeed, in order to recover from a single disk failure, we now need to read from many, if not most or all the disks in the disk array. However, the amount of data read from each disk is much smaller and our two layouts realize savings in the total amount of data read. For example, if only one disk has failed, then we only need to read from the data disklets in the same column for the rectangular layout and from the data disklets in the subgroup of the stripe in which the disklet is placed.

Unfortunately, modeling declustered disk arrays is difficult because the efficiency of declustering depends very much on the ratio of the total number of disks in the array over the total number of disks in the layout, so that an exhaustive investigation is difficult. Also, declustering algorithms are not easy to define and might not be very efficient. The size of the disklets is also important. If there are many disklets per disk, then the probability that a declustered array using one of our two layouts survives three failures is slim, however, if there are only 1000 disklets per disk, then the probability that all data survives three disk failures remains high. For example, the rectangular array with 5 rows and 20 columns survives three simultaneous failures with probability 0.999683267 and the same array (with 125 disks) well declustered survives with probability 0.728489, the corresponding Pyramid layout survives with probability 0.997781867 and the declustered version (also with 125 disks) with probability 0.108544. The declustered, rectangular layout is more robust than the declustered layout based on the basic Pyramid code. If however each disk consists of $100,000$ disklets, then the survival probabilities of the declustered layout go to $1.74695 \cdot 10^{-14}$ and $3.63673 \cdot 10^{-97}$. The rectangular layout is still spectacularly more robust than the Pyramid code layout, but this no longer matters.

In summary, declustered arrays are difficult to evaluate. As the disklet size shrinks, a declustered version of one of our layouts will suffer data loss with overwhelming probability with three failures, but never with two. The reliability of our two layouts becomes indistinguishable. The total hours of disks that have to be powered on for recovery however does not change significantly so that the ranking based on IO operations does not change, giving the advantage to the Pyramid code layout.

V. Modeling reliability and I/O recovery costs in the rectangular layout

A mathematically stringent determination of survival reliability and IO recovery cost for the rectangular layout is only possible for small number of failures. We fix a rectangular layout with r rows and s columns and assume that $r < s$. The array has rs data disks and $r+s$ parity disks for a total of $N = rs+r+s$ disks.

If there is only one failure, then dataloss is impossible. If the failed disk is a data disk or a column parity, then we need r reads in order to reconstruct the contents of the lost disk. If the failed disk is however a row parity, then s reads are necessary. The expected number of reads is

$$r\frac{s(r+1)}{rs+r+s} + s\frac{r}{rs+r+s}$$

If there are two failures, then dataloss is impossible. We need to distinguish a number of cases

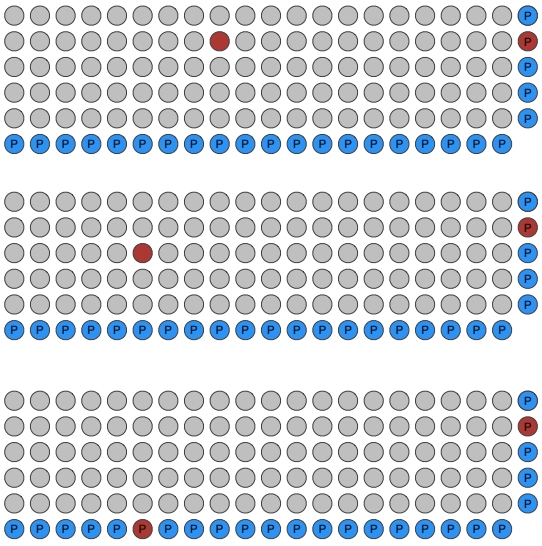

Fig. 6: Examples of a two-failure pattern in the rectangular $r \times s$ layout that can be repaired with $r+s$ reads. Failed disks are in red, parity disks in blue, and data disks in gray.

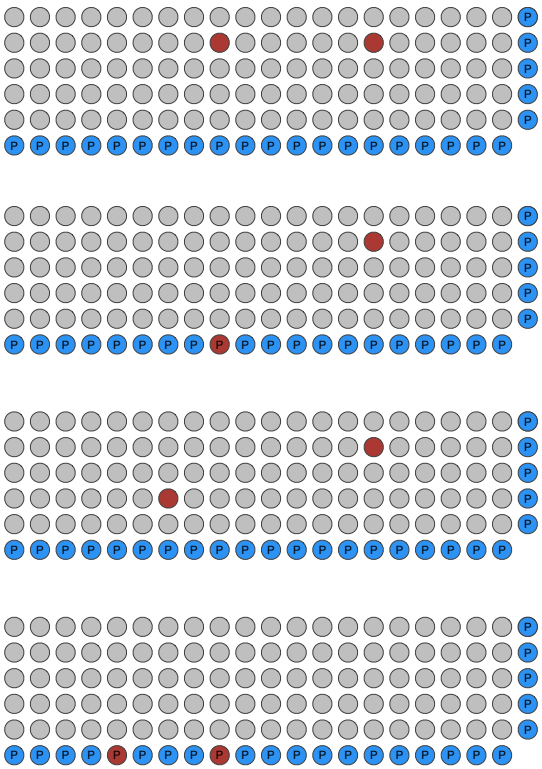

Fig. 7: Examples for the most common two-failure pattern in the rectangular $r \times s$ layout that can be repaired with $2r$ reads. Failed disks are in red, parity disks in blue, and data disks in gray.

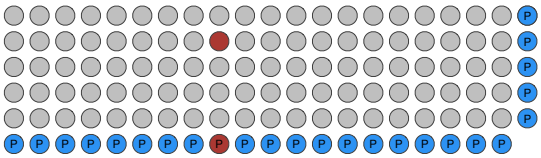

Fig. 8: Example for the two failures in the same row pattern in the rectangular $r \times s$ layout that can be repaired with $r+r$ reads. Failed disks are in red, parity disks in blue, and data disks in gray.

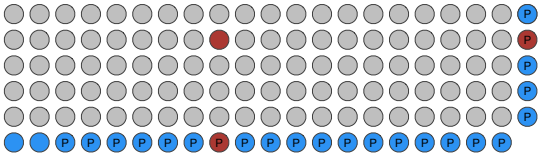

Fig. 9: Example for a three-failure pattern with dataloss in the rectangular layout. Failed disks are in red, parity disks in blue, and data disks in gray.

1) Both failed disks are row parities and we need $2s$ reads in order to recover. This happens with probability $\binom{r}{2}\binom{N}{2}^{-1}$.
2) One failed disk is a row parity and the other failed disk is a data disk in the same or in another row or the other failed disk is a column parity disk. Or easier said, the other failed disk is not a row parity. We repair the other disk's contents with r reads using the column stripe it is in and then repair the row parity disk with s reads. The probability for this to happen are $r(rs+s)\binom{N}{2}^{-1}$. We illustrate the subcases in Figure 6.
3) If none of the two failed disks are row parities and if they are not located in the same column, then we can recover with $2r$ reads, Figure 7. To count the number of cases, we select two columns and then one of the $r+1$ disks in the column. The probability of this case is therefore $\binom{s}{2}(r+1)^2\binom{N}{2}^{-1}$.
4) The two failed disks are located in the same column. Then we use a row to recover the contents of one disk and the column to recover the contents of the other one with $r+s$ reads. To count these cases, we select one column and then two disks in them, Figure 8. The probability of this case is therefore $s\binom{r+1}{2}\binom{N}{2}^{-1}$.

The expected number of disk reads in order to repair two failures is therefore

$$\frac{1}{2}rs\left(r^2(2s+1)+r(7s-1)+3s-4\right)\binom{N}{2}^{-1}.$$

If there are three failures, then dataloss is possible. Figure 9 shows the failure pattern. It consists of a failed data disk and failed parity disks in the same row and column. Since the pattern is uniquely characterized by the failed data disk, the probability of data loss are

$$rs\binom{N}{3}^{-1}$$

Using a similar set of case distinctions, we can also derive a formula for the expected number of disks to be read in order to recover from failure. This procedure can be continued for several numbers of failed disks but eventually runs into a combinatorial explosion of cases.

VI. Modeling RAID Level 6 / Pyramid reliability and I/O recovery costs

We calculate the survival probability of a modified RAID Level 6 consisting of a single stripe with m groups each encompassing n data disks each. Each group has a P (a.k.a. XOR) parity and each stripe has a Q parity.

Assume that f disks have failed. We distinguish three mutually exclusive cases in which no data has been lost:

(A) The Q parity has not failed and there is at most one failure in each group.
(B) The Q parity has not failed, there is one group with two failures and the remaining $f-2$ failures are afflicting different groups.
(C) The Q parity has failed and there is at most one failure in each group.

We calculate the number of different failure patterns. In case A, we select f of the m groups for containing a single failure and then we select one of the $n+1$ elements in each group for being the failed disk. This gives us

$$n_A(f) = \binom{m}{f}(n+1)^f$$

different failure problems. In case B, we select one group with two failures and then $f-2$ groups with one failure. This gives

$$n_B(f) = \binom{m-1}{f-2}m\binom{n+1}{2}(n+1)^{f-2}$$

different patterns. In case C, we select the $f-1$ groups with one failure among the m total groups.

$$n_C(f) = \binom{m}{f-1}(n+1)^{f-1}$$

patterns. Since there are

$$t(f) = \binom{m(n+1)+1}{f}$$

total failure patterns for f failures, the probability of all data to survive f failures (in a single stripe) is

$$\wp_1(f) = \frac{n_A(f)+n_B(f)+n_C(f)}{t(f)}.$$

We can also calculate the number of IO operations. In case A, we need to read n disks in each group with a failure and write to one spare, for a total of

$$r_A(f) = (n+1)f.$$

In case B, we need to read from $n-1$ disks in the group with the double failure, from n disks in each other group, and from the Q parity to recover the data from the two failed disks in the same group. This information is also sufficient to calculate the data on the other lost disks. Therefore, we have to read mn disks (corresponding to all data disks in our single stripe) and have to write to f for a total number of

$$r_B(f) = mn+f$$

IO-operations. In case C, we need to read mn disks in order to reconstruct the Q-parity. With this information we can also

recover from the other failures. Therefore we have

$$r_C(f) = mn + f$$

IO-operations.

We only count IO-operations if we do not have suffered dataloss. This means that the expected number of IO operations is

$$R_1(f) = \frac{r_A(f)n_A(f) + r_B(f)n_B(f) + r_C(f)n_C(f)}{n_A(f) + n_B(f) + n_C(f)}$$

If we have a RAID Level 6 organization based on the basic Pyramid code with several stripes, then we can calculate the probability of surviving f failures by combining the hypergeometric distribution with the results for a single stripe. Because we only count IO operations if we can recover successfully, extending these results to multiple stripes is difficult. In particular, the probability of having f failures in one stripe is not independent of the probability of data loss in one of the other stripes. We therefore had to take recourse to simulation for the RAID Level 6 Pyramid code organization as well.

VII. Results

We present now the results of our simulations. Each simulation run for f failures tells us whether data loss has occurred and what number of I/O operations were necessary to cover from failure if no data loss has occurred. The number of I/O operations is not normally distributed and it is not clear whether we can approximate it with a normal distribution. To obtain trustworthy results of the one-year survival rates, a plethora of individual runs were necessary. As many runs were already necessary, we determined the confidence intervals by grouping the runs into 30 subsets (batches) of 500000 individual runs. Since the results of each run are identically distributed, independent random variables, the central limit theorem applies and we can assume with exceedingly low error that the batch averages are normally distributed and obtain 95 percent confidence intervals that are very tight. In fact, our graphs cannot show error margins as they would be too small to see. Our simulation results also correlate well with the results of combinatorial calculations for small number f of failures.

The confidence intervals are typically in the range of $< \pm 0.1\%$ of the value. Only when we simulate high numbers of disk failures where the survival probability of the array data is less than 1% do we have noticeably higher error margins. Even then, the numbers for robustness and I/O operations for recovery can be compared with an exceedingly high statistical significance, since the confidence intervals are far from overlapping. The one-year data survival rates are quite sensitive to the robustness numbers for large mean times to failures of disks. But this does not pose a problem since the the survival probabilities for these high numbers of failures only contribute marginally to the annual survival rate.

The smoothness of the one-year survival graphs obtained is itself a testimony to the quality of the data we derived. While the simulation of the RAID Level 6 pyramid organization is reasonably fast, the one of the rectangular layouts is costly due to the need for minimizing the number of disks read. We used about 24 cores for almost two months. Using Python 3 as the programming language contributes to the excessive CPU use, but results in easier verifiable code. Extending the results to larger arrays would be costly. We did not try to simulate the presence of latent sector disks since any simulation of value would need to use a plethora of scenarios.

Figure 10 gives the survival probabilities for the rectangular layout and the RAID Level 6 variants as well as the difference. Each rectangular layout is identified by an r by s product where r is the number of rows and s is the number of columns for a total of rs data disks and $r+s$ parity disks. Similarly, each array organized as a RAID Level 6 array based on the Pyramid code with s stripes, each consisting of g groups with t data disks and $g+1$ parity disks is identified by $sxgxt$. This array has sgt data disks and $s(g+1)$ parity disks. The parity overhead of the rectangular and the Pyramid code based RAID Level 6 layouts are equal. Our figures compare arrays of equal size and parity overhead, as is indicated by the dashing of the curves.

As we can see, the rectangular layout is quite a bit more resilient. All layouts survive two failures for sure. It should be no surprise that larger layouts can sustain more disk failures than smaller ones, but if we look at the proportion of failed over total disks, then smaller layouts are more resilient, Figure 11. We can attribute this effect to the higher proportion of check disks in the disk population.

We observe that the robustness advantage for the rectangular layout improves if we increase the smaller group size r.

The reason that the rectangular layout is more robust is the better interconnectivity amongst drives. In the rectangular layout, the survival of a disk has a positive, though usually indirect effect on the probability that data on another disk survives. In contrast, different stripes in the RAID Level 6 organization are independent. With other words, a local clustering of errors in the rectangular layout is more likely to be survivable than in the RAID Level 6 pyramid layout.

This however means that the RAID Level 6 pyramid layout allows more localized recovery operations. As Figure 12 shows, the expected number of disks to be read per failed disk is substantially higher for the rectangular layout. Figure 12 depicts disk reads per failed disk repaired. If there are few failed disks, then we can most often chose an inexpensive way of data recovery. As however the number increases, we will be forced to include a more expensive recovery in the recovery plan. As the number of failed disks in the ensemble increases, we have to read close to the totality of disks so that the average starts sinking and becomes even lower than the number of reads needed for recovering a single disk. However, when this happens, dataloss becomes increasingly likely. Finally, we would like to remark that the most frequent case in a disk array with at best a few hundreds disks are none, one, two, or three failures at a time.

The superior robustness of the rectangular layout becomes more obvious if we look at the one-year survival probability

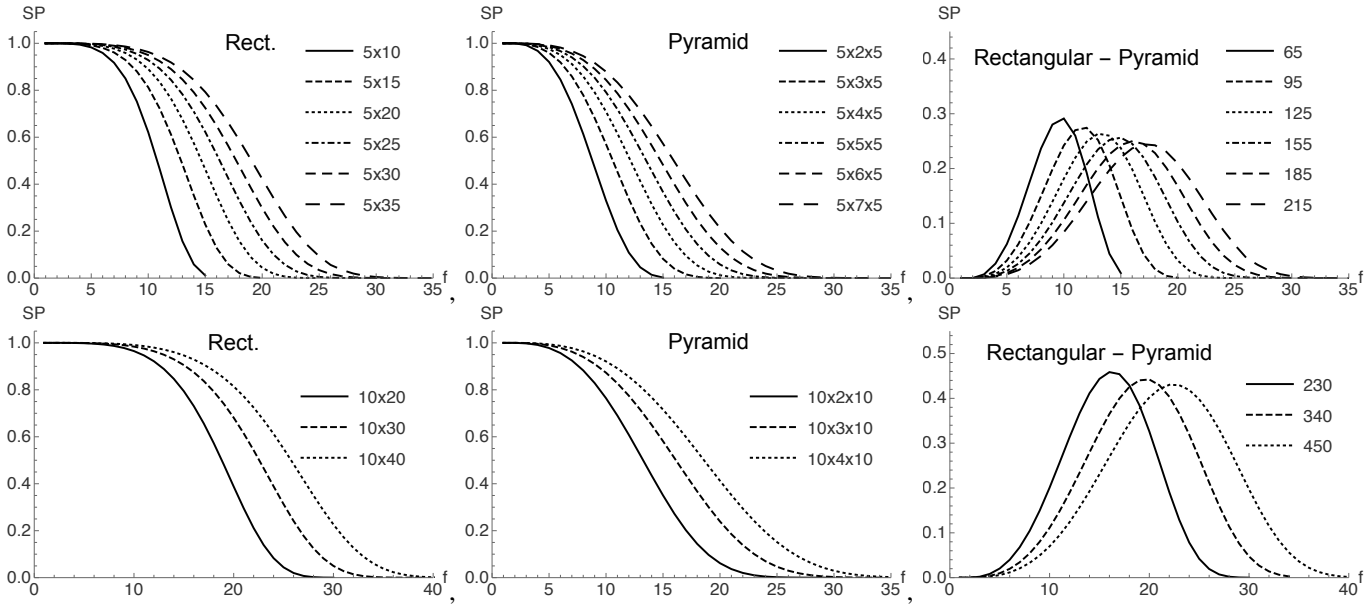

Fig. 10: Data Survival probability for the rectangular layout (left) and the Raid Level 6 variant (middle) with small group size 5 (top row) and 10 (bottom row) in dependence on the number f of failures. The right graph compares the layout by giving the difference between the survival probability of the rectangular layout and the RAID Level 6 variant.

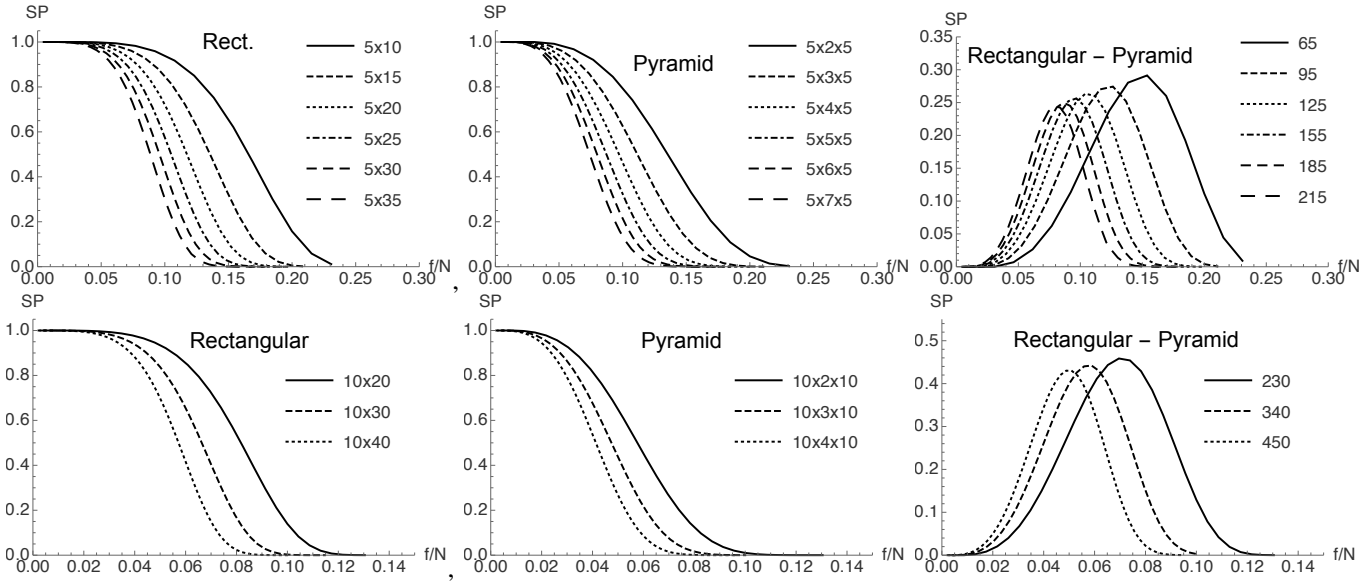

Fig. 11: Data survival probability in dependence on the proportion of failed disks over total disks for the rectangular layout (left) and the RAID Level 6 variant. The top row shows the results for the layouts with smaller group size 5 and the bottom for smaller group size 10.

defined to be the probability that no data loss has occurred in the first year. We model this with the standard Markov model, where State f corresponds to a state with f failed disks. The number of states is determined by the number of disk failures that can befall a certain disk array organization without leading to dataloss with overwhelming probability. An additional state is the failure state. We have two transitions. We denote the number of disks in the array with N. Let λ be the failure rate of individual disks, meaning that $1/\lambda$ is the average disk life expectancy. Let q_f be the probability that an additional failure in the presence of f already failed disks leads to dataloss. Thus, q_f is the conditional probability of dataloss with $f+1$ failures given that there was no dataloss with f failures. The values of the q_f are calculated from the absolute probability that f failures have produced dataloss. There is a failure transition from State f to State $f+1$ taken at rate $q_f(N-f)\lambda$. Second, we model repairs through an exponential repair rate ρ so that $1/\rho$ is the expected time to repair a certain disk. The repair transition from State f to State $f-1$ is taken with rate $f\rho$. Finally, there is a transition modeling dataloss

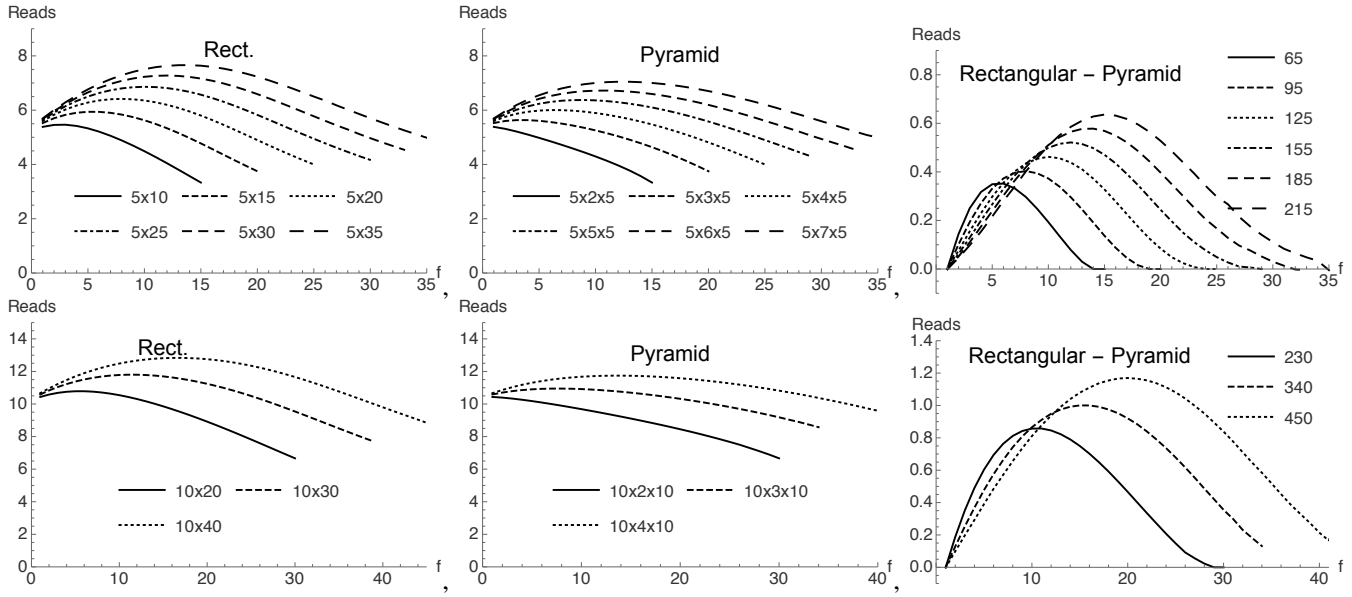

Fig. 12: Expected disks read for successful recovery in dependence on the number of failures. To the left are the rectangular layouts, in the middle the pyramid code layouts, and to the right the difference. The upper row are the numbers for smaller group size 5 and the lower row for smaller group size 10.

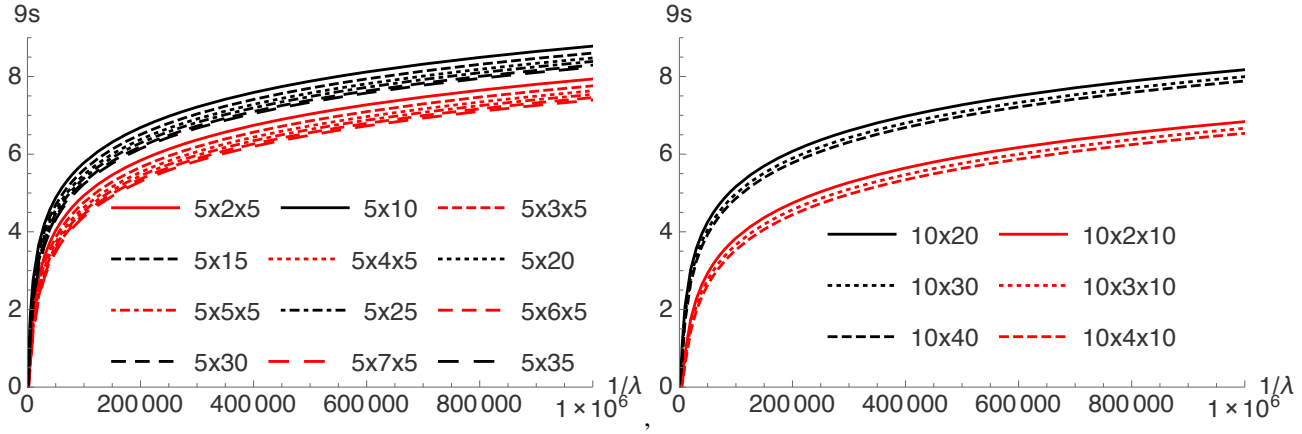

Fig. 13: One year survival probability (expressed in nines) in dependence on the average disk life expectancy expressed in hours.

from State f to the absorbing (failure) state at rate $(1-q_f)f\lambda$. Since exponential failure rates are somewhat unrealistic and since repair times certainly are not exponential, our results should be used for comparison purposes only. We used an average time to repair $1/\rho = 36$ hours, which is reasonably generous for an archival system and a tall order for a system that has lots of user data traffic. We assumed an unlimited number of spares and initial State 0, modeling the absence of failed disks.

We then used the Euler method to solve the Chapman-Kolmogorov differential equations corresponding to the Markov model. We express the resulting 1-year data survival rate in number of nines. Thus, a value of 4 corresponds to the chances of surviving one year of 0.9999. We give the results in Figure 13. As we can see, both layouts have survival rates that differ little. The figures for the rectangular and the Pyramid layout fall into narrow bands. The number of disks in a layout is less of a determinant of data survival rates than the layout chosen. The rates for the rectangular layout are consistently higher than the ones for the RAID Level 6 pyramid layout. The difference corresponds to the same reliability for double the disk failure rate of the rectangular layout compared to the pyramid layout. The comparisons are fair since we compare layouts with exactly the same number of data and of parity disks.

We finally notice that RAID Level 6 layouts can be configured in smaller increments. For example, we need five stripes of a layout with two groups of five data disks to obtain a layout with the same number of data and parity disks as the rectangular design with five rows and ten columns. This design has 50 data disks, but the RAID layout can be organized in increments of ten data disks, i.e. with 10, 20, 30, ... data disks.

We can summarize our findings as follows:

1) Rectangular layouts are considerably more reliable than RAID Level 6 layouts based on a simple pyramid code.
2) RAID Level 6 layouts based on pyramid codes have lower I/O rates than corresponding rectangular layouts.
3) RAID Level 6 layouts based on pyramid codes allow more flexible configurations.

VIII. Related Work

Greenan, Li, and Wylie [7] investigate storage systems laid out using flat XOR-codes with higher degrees of fault tolerance.

Erasure codes with lowered IO costs for recovery are not only attractive in archival storage systems, but also of interest in networked storage, where this feature also lowers the bandwidth of degraded reads. Khan et alinvestigate the application of MDS codes to networked storage and call for a more thorough investigation of the tradeoffs between storage efficiency, fault-tolerance, and performance [11]. Sathiamoorthy and colleagues describe a system with higher failure tolerance for use at Facebook's distributed storage system [14]. The resulting code is an extension of the basic pyramid code that we utilized. Their evaluation is a bit more applied than ours relying on measurements in actual clusters, but their criteria are similar. Our results are applicable over a wider set of uses.

Papailiopoulos and Dimakis and then Tamo and colleagues, Silberstein and colleagues, Cadambe and colleagues among others investigate codes that they call locally repairable codes [12], [3], [18], [20]. These are codes that – as their name suggests – are very efficient in reconstructing single disk failures. Network coding is an active area of research in codes [4].

IX. Conclusion

We have compared two different layouts that permit fewer reads on average for recovery of data on lost disk drives than comparable, more traditional layouts such as the two-dimensional or the RAID Level 6 layout. The rectangular layout only uses exclusive-or operations and shows superior resilience, but has slightly higher reads necessary during recovery. The layout based on the basic pyramid code needs Galois field operations to calculate the Q-parity, has lower resilience, but slightly lower average number of reads necessary for the recovery of data on failed disks.

In an archival system, the presence of large reliability stripes does not constitute an operational disadvantage. Only if the system experiences many updates would the parity disks belonging to such a long stripe see much traffic. Thus, we conclude that the energy savings of maintaining archival data on disk arrays based on the two investigated layouts come at no cost and we recommend strongly their adaptation for two-failure tolerant layouts.

References

[1] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, no. 1. ACM, 2007, pp. 289–300.

[2] B. Beach, "What hard drive should I buy?" www.backblaze.com/blog/best-hard-drive/, Backblaze, 2015.

[3] V. Cadambe and A. Mazumdar, "An upper bound on the size of locally recoverable codes," in *Network Coding (NetCod), 2013 International Symposium on*. IEEE, 2013, pp. 1–5.

[4] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Information Theory, IEEE Transactions on*, vol. 56, no. 9, pp. 4539–4551, 2010.

[5] J. G. Elerath and S. Shah, "Server class disk drives: how reliable are they?" in *Reliability and Maintainability, 2004 Annual Symposium-RAMS*. IEEE, 2004, pp. 151–156.

[6] J. Gray and C. Van Ingen, "Empirical measurements of disk failure rates and error rates," *arXiv preprint cs/0701166*, 2007.

[7] K. M. Greenan, X. Li, and J. J. Wylie, "Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–14.

[8] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, and D. A. Patterson, "Coding techniques for handling failures in large disk arrays," *Algorithmica*, vol. 12, no. 2-3, pp. 182–208, 1994.

[9] C. Huang, M. Chen, and J. Li, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," in *Network Computing and Applications, 2007. NCA 2007. Sixth IEEE International Symposium on*. IEEE, 2007, pp. 79–86.

[10] ——, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," *Trans. Storage*, vol. 9, no. 1, pp. 3:1–3:28, Mar. 2013. [Online]. Available: http://doi.acm.org.libproxy.scu.edu/10.1145/2435204.2435207

[11] O. Khan, R. Burns, J. Plank, and C. Huang, "In search of i/o-optimal recovery from disk failures," in *Proceedings of the 3rd USENIX conference on Hot topics in storage and file systems*. USENIX Association, 2011, pp. 6–6.

[12] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," in *information theory proceedings (ISIT), 2012 IEEE international symposium on*. IEEE, 2012, pp. 2771–2775.

[13] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population." in *FAST*, vol. 7, 2007, pp. 17–23.

[14] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," in *Proceedings of the VLDB Endowment*, vol. 6, no. 5. VLDB Endowment, 2013, pp. 325–336.

[15] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you?" in *FAST*, vol. 7, 2007, pp. 1–16.

[16] T. Schwarz, M. Baker, S. Bassi, B. Baumgart, W. Flagg, C. van Ingen, K. Joste, M. Manasse, and M. Shah, "Disk failure investigations at the internet archive," in *Work-in-Progess session, NASA/IEEE Conference on Mass Storage Systems and Technologies (MSST2006)*, 2006.

[17] T. J. Schwarz, Q. Xin, E. L. Miller, D. D. Long, A. Hospodor, and S. Ng, "Disk scrubbing in large archival storage systems," in *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings. The IEEE Computer Society's 12th Annual International Symposium on*. IEEE, 2004, pp. 409–418.

[18] N. Silberstein, A. S. Rawat, O. O. Koyluoglu, and S. Vishwanath, "Optimal locally repairable codes via rank-metric codes," in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 1819–1823.

[19] N. Talagala and D. Patterson, *An analysis of error behavior in a large storage system*. Technical Report CSD-99-1042, University of California, Berkeley, 1999.

[20] I. Tamo, D. S. Papailiopoulos, and A. G. Dimakis, "Optimal locally repairable codes and connections to matroid theory," in *Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 1814–1818.

[21] Q. Xin, E. L. Miller, T. Schwarz, D. D. Long, S. A. Brandt, and W. Litwin, "Reliability mechanisms for very large storage systems," in *Mass Storage Systems and Technologies, 2003.(MSST 2003). Proceedings. 20th IEEE/11th NASA Goddard Conference on*. IEEE, 2003, pp. 146–156.