# RESAR: Reliable Storage at Exabyte Scale

Thomas Schwarz, SJ*, Ahmed Amer†, Thomas Kroeger‡, Ethan Miller§, Darrell Long§, Jehan-François Pâris¶
*Marquette University, Milwaukee, WI, tschwarz@jesuits.org
†Santa Clara University, Santa Clara, CA, aamer@scu.edu
‡Sandia National Laboratories, Livermore, CA,[1] tmkroeg@sandia.gov
§SSRC, University of California, Santa Cruz, CA, {elm, darrell}@soe.ucsc.edu
¶University of Houston, Houston, TX, jfparis@uh.edu

*Abstract*—**Stored data needs to be protected against device failure and irrecoverable sector read errors, yet doing so at exabyte scale can be challenging given the large number of failures that must be handled. We have developed RESAR (Robust, Efficient, Scalable, Autonomous, Reliable) storage, an approach to storage system redundancy that only uses XOR-based parity and employs a graph to lay out data and parity. The RESAR layout offers greater robustness and higher flexibility for repair at the same overhead as a declustered version of RAID 6. For instance, a RESAR-based layout with 16 data disklets per stripe has about 50 times lower probability of suffering data loss in the presence of a fixed number of failures than a corresponding RAID 6 organization. RESAR uses a layer of virtual storage elements to achieve better manageability, a broader potential for energy savings, as well as easier adoption of heterogeneous storage devices.**

## I. Introduction

Large data centers containing exabytes of data on millions of devices present several important challenges. First, device failures and latent sector failure become frequent. Second, it is difficult to administer dynamic, heterogeneous systems, as such large systems must be. To address these challenges, we developed RESAR (Robust, Efficient, Scalable, Autonomous, Reliable) storage, which provides the same two-failure tolerance as RAID 6, but only uses exclusive-or (XOR) operations, with the resulting layout weaving through the storage system. Like EvenOdd, X-codes or row-diagonal parity codes, the RESAR layout avoids the need for advanced, but power-hungry processors to perform Galois field-based parity calculations. RESAR accomplishes this goal using a virtual layer that simplifies the management of recovery operations, allows for adjusting stripe sizes, and reduces the complexity of administering churn in the data center caused by individual device failure and decommissioning as well as the introduction of new batches of storage devices.

Ultimately, the value of data, the frequency of storage device failures, and the size of the installation determine the level of failure tolerance needed. The consensus for large storage systems is that two-failure tolerance is the right level of tolerance, assuming that data stripes are not large. We need two-failure tolerance because failure of a complete disk can uncover latent sector failure—failure of a single or a few sectors that can only be discovered by trying to read an affected sector—when trying to recover from a single disk failure. General data rarely demand greater failure tolerance if recovery is sufficiently fast. This greater failure tolerance can be achieved by using stronger error-correcting codes, where each data block is covered by more than two parity blocks, but one can also use mechanisms such as disk scrubbing [26] and disk health monitoring [15] for disk-based storage systems to detect failures before they can do harm. Techniques such as wear leveling [5] and the selection of the error-correction code in flash memory pages [7] can control component failure rates for flash-based systems. RESAR, as we present it here, only aims to provide two-failure tolerance, but more important data can always be protected by additional redundancy such as mirroring to a geographically distinct site, or inter-site parity [4].

The RESAR code is like a grid that weaves through the disk array, and has superficial similarities to helical entanglement codes [8] or Weaver codes [11]. A storage system that discovers a device failure reacts by using the RESAR-provided redundancy data to recover from the failure by reconstructing the data from the lost device, segment, or page and storing the reconstructed data elsewhere in the system. This recovery procedure can generate a sudden load increase if complete devices are placed into the reliability stripes of an erasure correcting code. A common solution is declustering [1], [13], [31], where not complete devices, but only logically (in the case of SSD) or physically (in the case of disks) contiguous parts, called *disklets* are organized into reliability stripes. Now, if a storage device fails, the data in the (hundreds to tens of thousands) of disklets it houses is recovered by accessing data in each disklet's reliability stripe. Our disklet-to-disk mapping distributes the other disklets in the reliability stripes over many disks. The recovery work is then distributed over many disks and proceeds faster. Faster recovery shrinks the "window of vulnerability", starting with a failure and ending when all lost data is rebuilt, to a few minutes. It is also good for performance since the storage system spends less time in the degraded mode where recovery operations interfere with client reads.

The task of defining an organization of disklets in a storage system is complicated by the dynamic nature of large storage systems, where batches of new components are added and where components leave because of failure, decommissioning, or migration (*e.g.*, when disks are moved to a new rack). As in Ceph [28], we use a metadata server to manage the

[1] Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

mapping of disklets to disks. This metadata server mediates between a single, conceptual RESAR layout and the allocation of RESAR disklets to disklets in actual disks. The RESAR layout extends to any number of disklets, but only changes when we alter the number of disklets in a reliability stripe to change the parity overhead. The metadata server bases its allocation on the current location of disklets in disks and disks in racks, and reacts to changes in the physical environment by reassigning virtual disklets to physical disklets. This can be done, for example, to balance load, to create "cold" disks, or to reflect dependence on common components. In the case of failure, it recovers a conceptual disklet by reconstructing the data in the failed disklet and placing it in a new location.

The conceptual RESAR layout and its specific form has the following advantages:

1) RESAR can deal with very large disk arrays consisting of heterogeneous storage devices.
2) The metadata server can easily balance the load between already-present disks and a batch of new disks by moving physical disklets without changing the conceptual layout.
3) Recovery in RESAR is more flexible and can avoid interference with a read by switching recovery to the other stripe which contains the disklet.
4) Placing disklets into two different stripes with one parity disklet each protects data better than placing them in a single stripe with two parity disklets.
5) RESAR minimizes the number of disks currently powered-on for writing by making it effective to store data in a log, despite the complication of having to update parity in two different reliability stripes.
6) RESAR can efficiently change the size of reliability stripes, allowing dynamic trade-offs between parity overhead and the effort needed to recover from disk failures.
7) RESAR recovery from disk failure is on the order of minutes, compared to hours for a system composed of classical RAID 6 layouts.

We will first explain how the introduction of the virtual disklet layer simplifies management of storage systems. We next describe how to create RESAR layouts and assess RESAR's reliability and performance. We then compare RESAR to a declustered data layout that organizes disklets in reliability stripes of $k$ disklets and two additional parity disklets, showing that RESAR has both higher reliability and equal performance compared with RAID 6 layouts. We conclude by suggesting additional approaches that can build on our RESAR approach.

## II. Virtual Layouts

Large data centers are very dynamic. Every so often, a new batch of hardware is integrated into the system and constantly, disks and other components fail. In a large system many storage components share the same support structure. For example, disks might be arranged in enclosures and the enclosures themselves are located in racks. Failure within the same support structure is sometimes correlated, as exemplified by anecdotes where a change of enclosure lead to a dramatic lowering of disk failure rates. Sometimes, the whole support

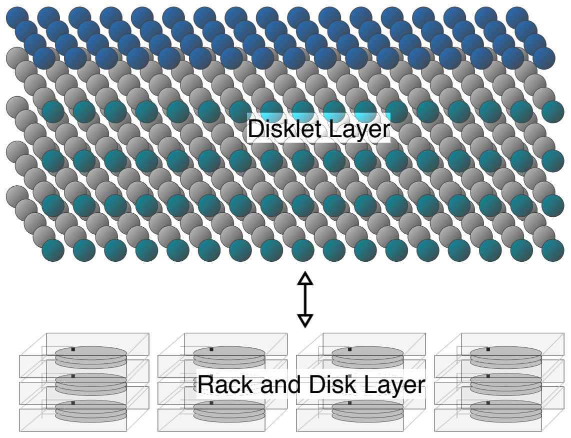

Fig. 1. A metadata server mediates between a virtual storage system consisting of data disklets (gray) and parity disklets (blue and cyan).

structure such as a rack can suffer a failure. For example, a network problem can disconnect all disks in one or more racks from the rest of the system.

Systems such as Ceph use a metadata server to administer this complexity [28]. We can use this technique to mediate between a virtual layout consisting of disklets arranged in reliability stripes (Fig. 1, top) and the actual hardware. A table maps disklets to "actually existing" disklets found in disks located in racks. In earlier work [6], we described this as *coloring* the virtual disklet with a disk and rack. The coloring needs to assign disklets in the same reliability stripe to different disks and possibly, to different racks. The name 'coloring' might be somewhat unfortunate since it suggests the complexity of the four-color theorem for graphs. In our case however, we have many choices for disks, enclosures, and racks so that the painting can actually be done *ad hoc*. To simplify coloring with racks, all disklets are numbered. If the number of two disklets differs by more than $R = k^2 - 1$, then they cannot share a reliability stripe and can therefore be placed in the same rack.

We therefore distinguish between a virtual disklet layer and a physical disklet layer. The virtual disklet layer is concerned with placing disklets into reliability stripes, while the physical disklet layer is concerned with placing disklets into disks located in enclosures and racks. Recovery from device failure is implemented in both layers, but at each layer the focus is on a more manageable and focused part of the data placement problem. The virtual layer is concerned with identifying the logical disklets needed to reconstruct unavailable virtual disklets, while the physical layer is concerned with the mapping between logical disklets and physical devices. As such, the virtual layer tackles redundancy management and recovery planning, while the physical layer focuses on the physical assignment of logical disklets to physical devices. Naturally there is coordination between the two layers, as it would be necessary to consult the mapping of physical to virtual disklets in order to identify the set of virtual disklets that need to be recovered when a particular physical device (be it a disk, a rack, or even an individual sector) fails. The mapping also influences the selection of physical locations for virtual disklets, as it is important to avoid assignment of multiple virtual disklets within the same reliability stripe to the same physical device.

The physical layer gives a list of failed disklets to the virtual disklet layer, which then begins to plan recovery operations. It might query the physical disklet layer for the current availability of disklet layers which reflects the load of the disk. As the virtual layer assigns reads to disklets, the physical layer keeps track of the additional load at the disks in which the disklets are placed. However, even without this information, the virtual disk layer can plan recovery. In the case of a major failure, recovery often has to proceed in stages. For instance, in order to recover the data from a given disklet, we need to read all other disklets in one of the two stripes, but there might be missing disklets in both of them. However, these in turn might be recoverable. Often, there may be no choice involved in how to recover a particular disklet. In these complicated situations, the virtual disk layer can act without knowing, or caring, where the disklets are located. If, for example, the virtual layer has the opportunity to choose between two possible paths of recovering a particular data disklet, then more detailed information from the underlying physical layer may become more important. For example, it may be more beneficial to select a recovery path that involves the most underutilized physical devices as a means of load balancing. Or, in the interests of energy efficiency, it may be best to select the physical devices currently active, and thereby avoid spinning up idle hardware from a low power state. The option of making such a choice is an advantage of this virtualization of the disklets, and is increasingly likely to be useful as systems scale increases in the number of devices. If a disk fails, the disklets located in it are reconstructed in other disks,which is represented by repainting the virtual disklets. This operation leaves the virtual disklet layer intact and thus does not affect the definition of the reliability stripes.

The introduction of the virtual disklet layer also simplifies manipulation of the storage system, which offers benefits in routine management even in the absence of failures. If the system needs to rebalance load, for example after incorporating a batch of new disks, data needs to be copied from a disk to another disk, but the virtual disklet layer does not need to change.

If we want to make trade-offs between the failure tolerance of the array and the storage overhead, we need to change the parameter $k$ giving the number of data disklets in each reliability stripe. For example, if we want to halve $k$, we have to break reliability stripes in half and then add an additional parity drive. This is an operation solely in the virtual disklet layer, which is easy to perform. The result is a generalized RESAR layout, until the disks are renumbered. However, no actual data or parity disklets need to change place. Similar operations are possible to double, increment, or decrement $k$.

In summary, the introduction of the virtual disk layer reduces the administrative complexity of dealing with failure and growth.

## III. Two-Failure Tolerant Flat XOR Codes

The underlying code for RESAR is a flat XOR-code. Greenan *et al.* defined them as codes where parity symbols are calculated from certain subsets of data symbols with an exclusive-or operation [10]. We call these subsets *reliability stripes*. Each flat XOR-code corresponds directly to a disk array layout where each data symbol corresponds to a data disk and each parity symbol to a parity disk.

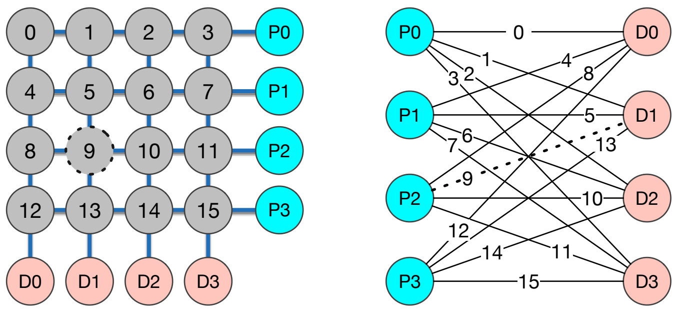

Fig. 2. Left: The two-dimensional layout for a disk array. 0, 1, 2, . . ., 15 are data disks and $P0$, . . ., $P3$ and $D0$, . . ., $D3$ the two set of parity disks. Right: The corresponding graph visualization, where edges correspond to data disks. For example, data disk 9 located in the stripes with parities $P2$ and $D1$ corresponds to the edge between $D1$ and $P2$ on the right.

Layouts based on flat XOR-codes that tolerate two simultaneous failures need to place each data disk in at least two different reliability stripes. The intersection of two reliability stripes cannot contain more than a single disklet. We can *label* each data disk by the numbers of the two reliability stripes to which it belongs. Similarly, we can label each parity disk with the number of the reliability stripe to which it belongs. This defines a graph structure derived from the disk layout where each parity disk corresponds to a vertex and each data disk to an edge between vertices.

We use the graph to visualize the layout of disklets into reliability stripes and later manipulate them. Fig. 2 gives an example. In the graph layout, parity disklets corresponds to vertices and data disklets to edges. To our knowledge, the graph visualization was first exploited by Xu *et al.* in the definition of B-codes [32], as precursor to Row-Diagonal parity codes that only uses exclusive-or operations and gives two failure tolerance. An observation by Zhou *et al.* characterizes minimal failure sets of disks as those containing either a cycle of edges or a path where the end vertices have also failed [35]. The graph visualization is a good way to determine the failure resilience of these type of layouts [23].

We base our RESAR layouts on a bipartite graph. RESAR layouts are scalable and can incorporate an arbitrarily large number of data disklets (or complete storage devices). In the graph presentation, the layout consists of two columns of parity disklets, which we call the P (parity) and the D (Diagonal parity) disklets (Fig. 3). Each P-parity disklet is "connected" (by a data disklet) to its opposite D-disklet and

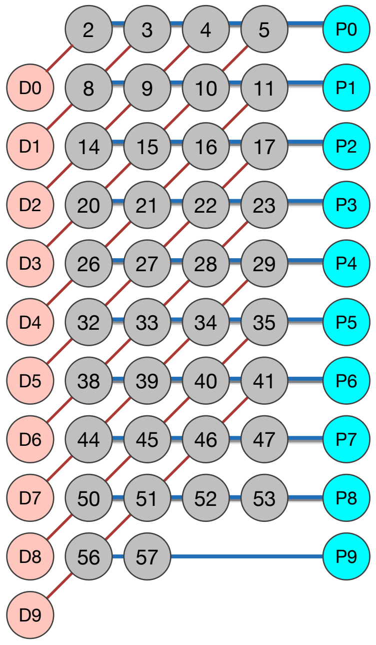
Fig. 3. A small bipartite RESAR layout with $k = 4$.

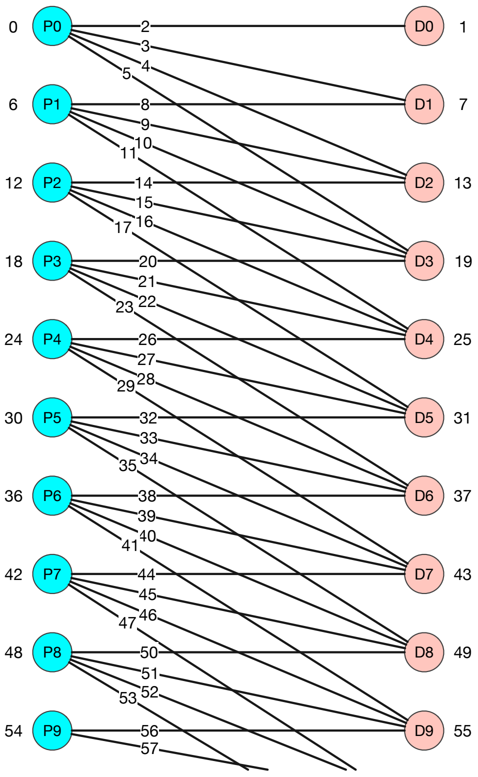
Fig. 4. The graph representation of the RESAR layout in Fig. 3.

the next $k - 1$ D-disklets (where $k$ is the number of data disklets in a stripe). If we so desire, we can have the layout loop around so that all reliability stripes encompass exactly $k$ data disklets. This however would make it more difficult to add additional disklets to the ensemble.

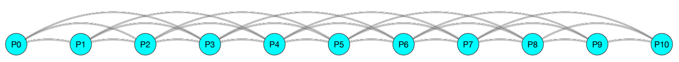
Fig. 5. A small complete RESAR layout in graph form.

Fig. 3 gives a small RESAR layout (with $k = 4$) on the left and the graph representation of the layout on the right. In the layout, the disklets are arranged in rows of $k$ disklets with a $P$-parity added to it. The rows are then arranged in columns and each disklet becomes part of a diagonal stripe with $k$ data disklets and an additional $D$-parity.

We use an enumeration of disklets not only to indicate how the disk array would grow if more disks are added, but also to control the placement of disklets in disks such that a single rack failure cannot cause data loss. If the elements in Fig. 3 denote disklets and we place disklets 33, 35, 38, and 40 in the same disk, then the failure of this disk causes data loss, since all four disklets share their two reliability stripes with another one of the four. Similarly, a rack failure can cause data loss if we place these disklets in different disks, but in the same rack.

Our enumeration proceeds in two stages. First, we number all disklets in a RESAR layout by a pair of numbers $(i, j)$ where $i$ enumerates consecutively the $P$-parities and $j, 0 \leq j \leq k+1$. The $P$-parity gets $i = 0$, the opposite $D$-parity $i = 1$ and the data disks in the $P$-reliability stripe values $i = 2, 3, \ldots, k+1$ (Fig. 3). In the second stage, we convert the pair of numbers to a single value by the formula $(i, j) \rightarrow i \times (k+2) + j$. For example, the data disk 39 lies in the parity stripes of $P6$ and $D7$. It gets index $(6, 3)$ in the first stage and $6 \times 6 + 3$ in the second stage. As we will see, failure patterns are made up of unavailable disks (or disklets) that are made up of neighboring elements in the graph. The indices of the data disks in the stripe of $Dj$ are $(j - l, l + 2) \rightarrow (j - l((k+2) + (l+2))$ for $0 \leq l \leq k$. The biggest difference between the indices of neighboring disks is attained when both are data disks sharing a stripe with a $D$-parity $Dj$. It is between disks $(j - k + 1, k + 1)$ and $(j, 2)$, *i.e.*, between $(j-k+1)(k+2)+(k+1)$ and $j(k+2)+2$ and is $R = k^2 - 1$. If we have $R+1$ or more racks and assign disklets in a round-robin manner, then two disklets in the same rack can never be neighbors. This guarantees that a rack-failure alone can never cause data loss. The Python code that implements this layout is:

```
def diskNumber(i,j):
     return i*(k+2)+j
def pair(diskNumber):
     return (diskNumber //(k+2), diskNumber % (k+2))
def horizontalParity(diskNumber):
     (i,j) = pair(diskNumber)
     return i
def diagonalParity(diskNumber):
     (a,b) = pair(diskNumber)
     return a+b-2
```

In order to show the flexibility of RESAR layouts, we give another layout in graph form in Fig. 5. The design consists of a long sequence of parity disks $p_0, p_1, p_2, \ldots, p_n$. The reliability

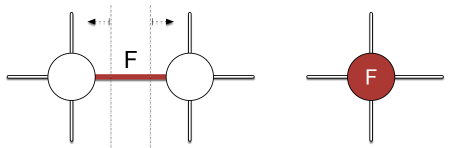

Fig. 6. Left: Failed data disklets with its neighbors. Right: Failed parity disklet with its neighbors.

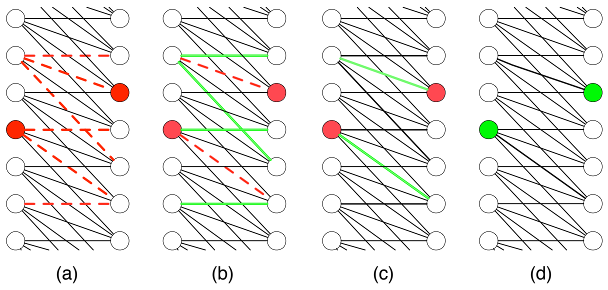

Fig. 7. Cascading recovery of multiple failures in RESAR.

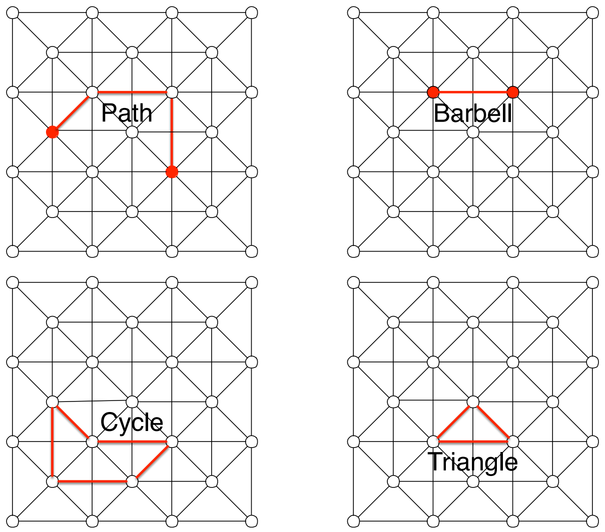

Fig. 8. Irreducible failure patterns. The barbell and the triangle on the right are irreducible failure patterns with the minimum number of failed disklets, namely three.

stripe with parity disk $p_i$ consists of $k = 2l$ data disks, which are located in additional, different reliability stripes with parity disks $p_{i-l}, p_{i-l+1}, \ldots, p_{i-1}, p_{i+1}, p_{i+2}, \ldots, p_{i+l}$. Reliability groups at the beginning and at the end of the sequence of course are smaller.

## IV. RESAR's Disk Failure Recovery

To recover the data from a failed drive we must recover the data in each of the disklets in the drive. Since each data disklet is in two reliability stripes (see Fig. 6 for a depiction in the graph visualization), we can recover using either reliability stripe if all the other data disklets and the parity disklet in the corresponding stripe are still available. This flexibility is useful to avoid using a heavily loaded disk for recovery and is also a reason for the greater robustness of this layout. A parity disklet on a failed disk can be reconstructed in another disk if all its data disklets can be recovered. RESAR can use both stripes to halve the reconstruction time.

We use the graph visualization to discuss recovery. The graph represents disklets, not disks, and a single disk failure results in multiple disklet failures. As we mentioned, a good disklet-to-disks mapping ensures that these disklet failures resulting from the failure of a single disk are widely spread over the (rather large) graph. We can recover the data from a disklet (and then place the recovered disklet on another disk drive) if it is represented by a vertex (and is therefore a parity disklet) if all the edges (data disklets) adjacent to it are available. We can recover the data from a disklet represented by an edge (*i.e.* a data disklet) if one of the adjacent vertices and all the edges adjacent to it are available. Cascading recovery in a RESAR layout happens if a disklet can only be recovered after some of its neighbors have been recovered. Fig. 7 gives in column (a) a case with several failures in our graph layout. Failed elements are marked in red. In the upper failure cluster, there is one vertex on the left, which has lost three adjacent edges. For this reason, this vertex, or to be more precise, the reliability stripe represented by this vertex, cannot be used for recovery, but each data element in this stripe is also located in another reliability stripe. Two of the failed data elements can be recovered directly, see column (b), and the remaining failed element's data can be recovered in the third step, see column (c). The failed vertex in this cluster can only be reconstructed if all its adjacent edges are available. The lower failure cluster contains a failed vertex from which a path of failed edges emanates. This pattern resolves itself only in the fourth step.

Arguments about failure tolerance are made much easier in the graph visualization than in the layout itself, as was previously observed [35]. Disk and sector failure induce a *failure pattern* in the graph. We are especially interested in patterns that represent data loss and that are minimal in the sense that removing one element of the pattern yields a pattern of failure from which we can recover. Any failure pattern that implies data loss is or contains at least one minimal failure pattern. A key observation is that an edge, that is part of a minimal failure pattern, either has end-vertices that also have failed or an end-vertex where one other adjoining edge has also failed. This allows us to classify all minimal failure patterns. They form either a cycle consisting of failed edges, or a path that starts and ends at a failed vertex and otherwise consists of failed edges in between. The smallest minimal failure patterns are the *barbell* and the *triangle*. Fig. 8 illustrates these concepts.

## V. RESAR Reliability

We now compare the reliability of a RESAR layout with that of a layout that organizes all disklets in the stripes of a RAID Level 6 layout with two parity disklets per stripe and $k$ data disklets, Fig. 9. The parity overheads in both organizations are equal (namely $2/k$). The RAID Level 6 layout consists simply of a very large number of these stripes.

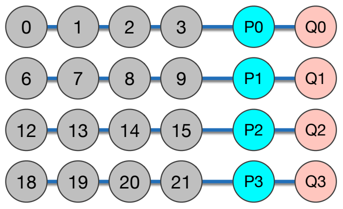

Fig. 9. Four RAID Level 6 reliability stripes.

Simulating reliability of even a moderate RESAR layout is involved, as instances of failure loss are so rare, even with a reasonably high number of failed disks. Therefore, we simulated a layout with 100,000 disks with single disklets over several months spending about 10000 CPU hours. We used these results in order to calculate the reliability of disk layouts with more disklets per disk. In order to do so, we had to make the following modeling assumptions. Of course, these assumptions negate the RESAR design philosophy and assumptions, for instance because an advantage of RESAR is the capability to deal with heterogeneous disk. An additional reason for these simplifications is that modeling heterogeneity is difficult, since modeling it involves making assumptions such as deciding the distribution of sizes and failure rate within the disk population.

1) All disks have the same number $N$ of disklets.
2) The disklets are organized in $N$ different cyclic RESAR layouts. The first disklet in a disk is in the first RESAR cycle, the second disklet in the second cycle, *etc.* The RAID Level 6 layout uses also $N$ different sets of reliability stripes and assigns a disklet in position $i$ in the disk to set $i$.
3) The assignment of disklet $i$ in a disk to a virtual disklet in the $i^{\text{th}}$ RESAR cycle or the RAID reliability stripe set is random and independent from the the assignment of the other disklets in the same disk.

We first determine data loss probabilities for non-declustered layouts, where there is one disklet per disk. We use combinatorics to count failure patterns based on the irreducible patterns such as those depicted in Fig. 8. This allows us to calculate the probability of data loss in a single RESAR cycle for less than seven failures. The derivations are too involved to present here. For the data loss probability of a RAID layout with $n$ disks, a stripe size of $k+2$, and $f$ failure and consequentially $s = n/(k+2)$ number of stripes, we count all ways of distributing the $f$ failures among the $s$ stripes such that no stripes has more than two failures. In the following formula for the probability of dataloss, index $i$ stands for the number of stripes with one failed element and $j$ for those with two failed elements.

$$1 - \left( \sum_{i\geq 0, j\geq 0, 2j+i=f} \binom{s}{i}\binom{s-i}{j}\binom{k+2}{1}^i\binom{k+2}{2}^j \right) \Big/ \binom{n}{f}.$$

Of course, both designs never suffer data loss with 2 or fewer failures. If $p_b(n,k)$ denotes the probability of data loss for a RESAR cycle with $n$ disklets and $k$ data disklets per stripe, then for $k \in \{3,4,5,6\}$

$$\rho = \lim_{n\to\infty} \frac{p_r(n,k)}{p_b(n,k)} = \frac{k(1+k)(4+k)}{6(2+k)},$$

which is 50.37 for $k = 16$ and 14.4 for $k = 8$. The non-declustered RESAR layout is $\rho$ times more robust than the non-declustered layout with RAID Level 6 stripes. We used extensive simulation (10000 CPU hours) to determine the data loss probabilities for a RESAR layout with 100,008 disks and $k = 16$ and up to 510 failures and found that the robustness of the RESAR layout was at the end only about 30 times better than the robustness of the RAID layout, as Fig. 11 shows.

If we move to a declustered layout, where there is more than one disklet per disk, this discrepancy remains, Fig. 12 and Fig. 11. Since our simulations did not yield very close confidence intervals (at the 99% level calculated using Wilson's formula [30]), Fig. 10, we use quadratic curve smoothing for the RESAR simulation results and exact values for the RAID layout. Fig. 12 and Fig. 11 gives the results when we place 100 disklets, 1000 disklets and 10,000 disklets per disk. With a 10 TB disk, this corresponds to disklets sizes of 100 GB, 10 GB and 1 GB. For the number $f$ of failed disks in the depicted range up to 20, the data loss probabilities for the declustered RAID are 50 times higher than for the corresponding RESAR layout.

Of course, we are really interested in reliability, measured for instance as the annual data loss rate or the annual data loss probability of a system. Unfortunately, we cannot do this calculation. First, both organizations can withstand a large number of disk failures, resulting in absorbing semi-Markov models that are for us impossible to evaluate or even to simulate. Modeling would also depend on the details of the layout. Second, system reliability depends on a large number of other factors such as the time to detect a failure, the occurrence of latent sector failures, storage load factor, reliability of racks, networking, and other shared components, as well as device characteristics such as the time needed to completely read a device. Modeling assumptions such as an unrealistically high rack failure rate can completely hide the difference in data layout, which we are trying to improve. The two factors that are different between a RESAR and a RAID 6 layout are the probability of incurring data loss after $f$ failures and the reconstruction speed. As we will describe in the next section, the RESAR layout is has not only a smaller probability of occurring data loss after $f$ failures, but can also reconstruct the vast majority of user data on the lost device almost twice as fast, because each data disklet is located in two otherwise disjoint stripes.

## VI. Reconstruction Speed

A declustered RAID 6 layout places two parity disklets in the same reliability stripe as $m$ data disklets. To reconstruct the contents of a disklet on a failed disk, the recovery process

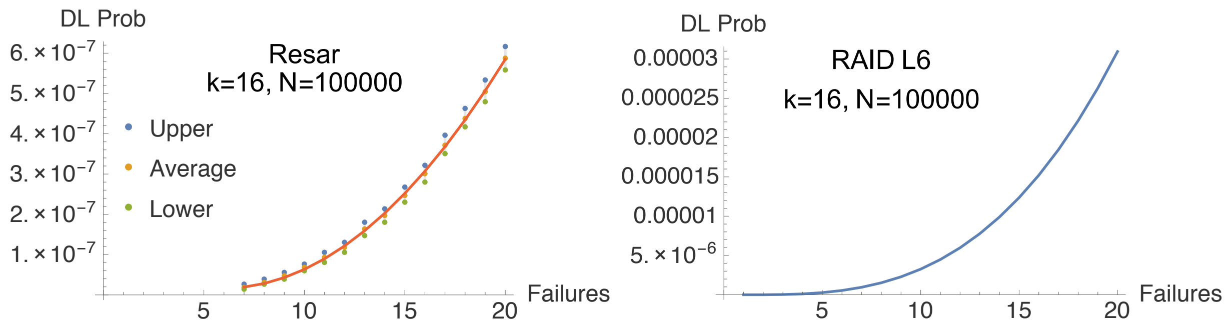

Fig. 10. Simulation results for the data loss probability (DL Prob) of a RESAR layout (left) and exact values for a declustered RAID Level 6 layout (right), both with 100,000 disks. On the left, we give the average and the upper and lower value for the 99% confidence interval calculated with Wilson's formula. The graph is that of a quadratic function in the number of failures that best fit the simulation results. Note the different scales for the $y$-axis.

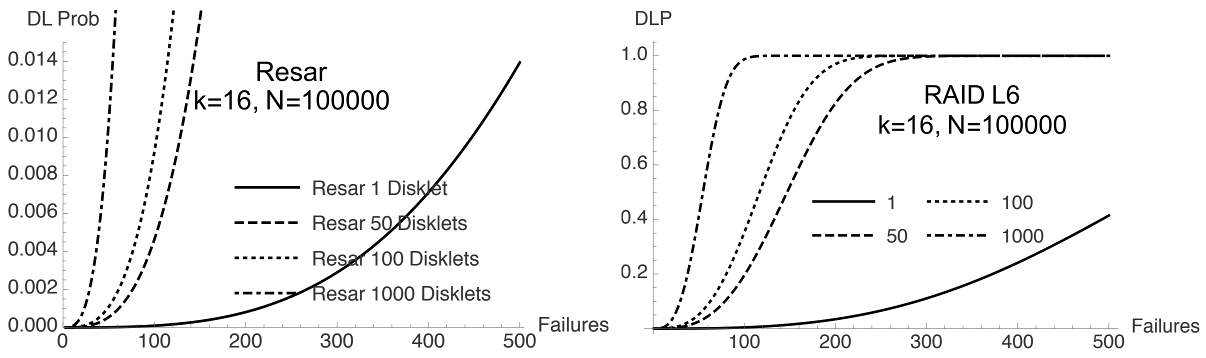

Fig. 11. Simulation results for the data loss probability (DL Prob) of a RESAR layout (left) and exact values for a RAID Level 6 layout (bottom) with 100,000 disks. On the left, we give the average and the upper and lower value for the 99% confidence interval. We also added two smoothed curves for the data loss probability if each disks in the RESAR layout carries 50, 100 and 1000 disklets respectively on the left and the data loss probabilities for the declustered RAID Level 6 layout with 1, 50, 100, and 1000 disklets. Note the different scales for the $y$-axis. Fig. 12 compares the systems directly.

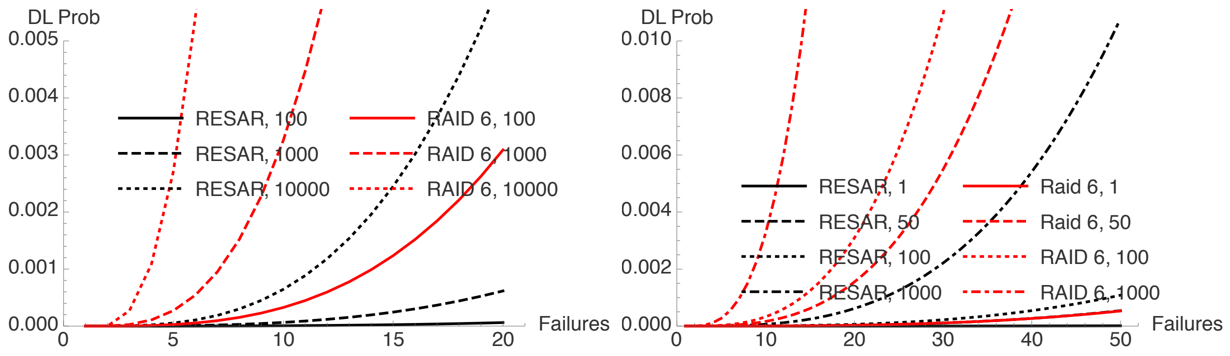

Fig. 12. Two comparisons of the data loss probabilities of RESAR and RAID 6 systems in Fig. 11. The system has 100,000 disks and various numbers of disklets per disk. The curves for RESAR with 1 and 50 disklets are too close to the x-axis to be seen as different.

needs to read $m$ out of the $m+1$ remaining disklets in the stripe. Instead of selecting $m$ fixed disks to read from, it is common practice to distribute the reads over all $m+1$ disks, lowering the amount to be read by a factor of $m/(m+1)$. If the recovery wants to avoid writing as a bottleneck, it can further improve the speed by storing parts of the reconstructed disklet temporarily elsewhere.

A RESAR layout places the two parity disklets covering a given data object into two different reliability stripes, giving us two independent opportunities to reconstruct. Both stripes have no disklet in common, allowing us to reconstruct one half of the failed disklet's contents using one stripe and the other one using the other stripe. One of the results of this recovery operation is written to the location where the replacement disklet is allocated, while the other result is stored in a temporary location and later written to complete the reconstructed disklet at the allocated site. This limits the window of vulnerability after a disk failure to the time to detection of a disk failure and about half the time needed to read a disklet.

If a disk fails, the recovery operation needs to recover the contents of *all* disklets on the failed disk. However, while the

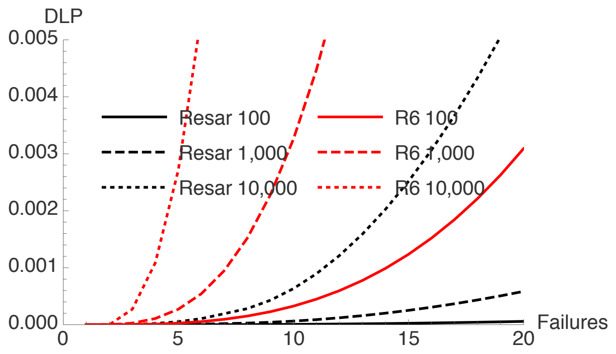

Fig. 13. Data loss probabilities for a system of 100,000 disks with various numbers of disklets per disk. This figure compares the curves in Fig. 11, but leaves out the curves for 1 and 50 disklets, since they are too close to the $x$-axis.

TABLE I
MAXIMUM ENTANGLEMENT NUMBERS IN A SYSTEM OF 1 MILLION DISKS USING A STRIPE SIZE OF 8. WE VARY THE NUMBER OF DISKLETS PER DISK.

| **Max. Number** | **Probability** | **95% conf. interval** |
|---|---|---|
| 1000 disklets per disk | | |
| 2 | 50.911% | ±0.191% |
| 3 | 48.814% | ±0.119% |
| 4 | 2.727% | ±0.018% |
| 5 | 0.002% | ±0.001% |
| 2000 disklets per disk | | |
| 2 | 0.504% | ±0.028% |
| 3 | 95.234% | ±0.065% |
| 4 | 4.235% | ±0.056% |
| 5 | 0.028% | ±0.007% |
| 5000 disklets per disk | | |
| 3 | 20.173% | ±0.192% |
| 4 | 77.335% | ±0.172% |
| 5 | 2.458% | ±0.075% |
| 6 | 0.035% | ±0.007% |
| 10000 disklets per disk | | |
| 4 | 46.648% | ±0.221% |
| 5 | 51.355% | ±0.203% |
| 6 | 1.956% | ±0.051% |
| 7 | 0.040% | ±0.007% |
| 8 | 0.002% | ±0.002% |

disklets that can be used for recovery of a single disklet are guaranteed to be on different physical disks, this is not the case for all disklets. If $A$ and $B$ are two storage devices, we define the *entanglement* of $A$ and $B$ to be number of disklets $D$ on $A$ such that there is a disklet $D'$ on $B$ that is a member of a reliability stripe containing $D$. If all entanglement numbers are zero or one, then recovery from a single disk failure in RESAR can be as fast as the time necessary to read half of a disklet. If an entanglement number of a failed disk is 2, then complete recovery takes twice that time, though of course the vast majority of recovery is done in half the time. Higher entanglement numbers do not mean that the recovery process has to take proportionally more time, since in the vast majority of cases, we can forego using one recovery stripe for reconstruction in order to avoid sending another read request to the same disk.

It is possible to keep track of entanglement numbers when creating the layout, but in a large storage system with $N$ devices, this means maintaining an entanglement number for $N(N-1)/2$ pairs of devices. We now investigate whether this is a useful enhancement to RESAR by assuming that entanglement is not considered in the layout. We determined the entanglement numbers for a system with 1 million disks and a reliability stripe length of 8 data and one parity disklet and give the corresponding probabilities for maximum entanglement in Table I.

We determined by simulation of a layout with 1000 disklets per disk, that about 98.41% of the $2 \times 8 \times 1000$ disklet halves to be read are the only disklet on the disk, that 0.79% of disks to be read harbor two disklet halves to be read, 0.004% of disks to be read harbor three disklet halves to be read and $1.8 \times 10^{-5}$% harbor four disklet halves. This means that the vast majority of contents are recovered very quickly.

## VII. Performance

RESAR is optimized for writes to consecutive data disklets. The need arises because we want to organize writes such that we write all data disklets in a reliability stripe and then update the parity disklet directly. This avoids the “small write” penalty. With the recent switch to shingled write disks and the need to save energy by powering off disks, data centers have new incentives to organize their data into several or even numerous logs. If they do so, writing to consecutive data disklets becomes the norm.

In the RAID Level 6 layout, there is only a single reliability stripe and we need to store the contents of $k$ data disklets “in flux” before we can calculate the parities. Alternatively, we can create a buffer for each of the two parity disklets and maintain the parity of the data disklets in the stripe already written in it.

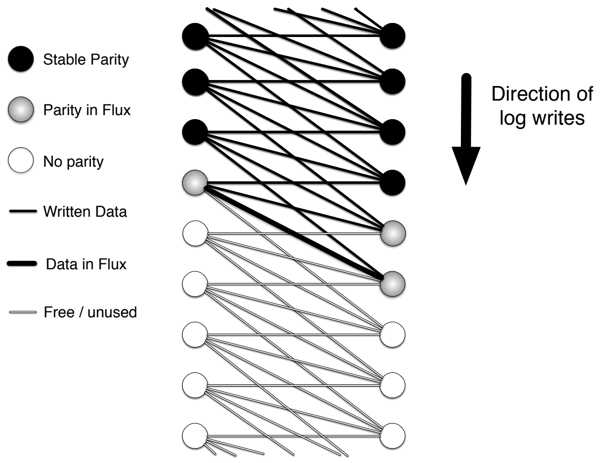

Fig. 14. Log write operation in a RESAR layout (with $k = 4$).

If we are writing a data disklet in a flat XOR-code, we can then write another data disklet in the same reliability stripe, but this leaves $k-1$ data disklets in the other reliability stripe, in which the original data disklet was, unwritten. As Fig. 14 shows, the bipartite RESAR layout actually deals quite well with this problem. We write data disklets in order of their indices. A parity disklet is in flux, if some, but not all the adjoining data disklets are in flux. In Fig. 14, there are three parity disklets in flux, one $P$-disklet and two $D$-disklets (the gray vertices on the right). If we write the next data disklets, one more $D$ disklet would be in flux, but if we then move on to the next data disklet, one $D$ disklet can be closed and no additional one enters a state of flux.

Since we are writing data disklets one at a time, only one data disklet (per log) can ever be in flux. Unlike for the RAID Level 6 layout, more than two parity disklets are in flux, namely a maximum of $k + 1$.

Dealing with a large number of parity disklets in flux is in fact not difficult. The simplest mode is to store copies of data disklets contents in other disks until all data disklets in a reliability stripe have been written. At this moment, we can calculate the parity data, place it in a parity disklet and then free the disklets with the content copies. We do not need to keep the disks with the data disklets replica powered on, but only need to power them on if they are needed, first to store the replicated data, and then in order to be read for the parity calculation.

The difference in processing writes between the two data layouts (RESAR and RAID Level 6) only lies in the additional number of disklets to be stored, which is very small compared to the total capacity of disklets and costs less than the price of a single hard drive. The RAID Level 6 layout uses up to $2k$ disklets for temporary copies of data, while the RESAR layout would use up to $k(k + 1)$ temporary disklets.

Most of the time, we would write to several data disklets in parallel in order to digest the amount of incoming data. The small advantage of the RAID Level 6 layout due to its compactness would then be even smaller.

## VIII. Related Work

Virtualization is of course a standard method to administer and organize computer systems and data storage is no exception. The introduction of a virtual layer as presented here appears to be novel. We now review some other uses of virtualization.

Data centers gain agility by integrating server and storage virtualization. Storage virtualization abstracts physical storage to present virtual disks to applications and individual users in a similar manner that server virtualization decouples virtual machines from underlying physical machines [20]. Harmony by Singh *et al.* is an early example of integrating both types of virtualization. But virtualization by itself is an effective tool to distribute load over several physical disks while maintaining quality of service guarantees [27]. Façade presents a small disk array to applications as a collection of virtual disk and allows thereby the establishment of performance guarantees [18]. Zhang and colleagues [34] present AVATAR, a low-level feedback-driven request scheduler that allows to meet service-level objectives for a large number of users that are isolated from each other and therefore can interact with individual virtual storage systems that share the same hardware. There is much additional work on how to isolate performance for different applications and users [3], [16], [17]. The same virtualization technology becomes even more attractive when we move from individual data centers to cloud storage [17], [33].

In this article, we consider a very different type of virtualization, whose purpose is not agility in order to adhere more strictly to service level guarantees, but agility in the reliability layout of large disk (including SSD) farms. Related work in disk array layout comes mainly from the nineties. A big difference to the current work is between homogeneous and inhomogeneous disk arrays and the size of the disk array. For example, DATUM by Alvarez, Burkhard, and Christian, only handles small numbers of disks [1]. A large number of contributions has been made to optimize disk array layouts for various configurations [2], [14], [21], [22], [24], [25].

In order to administer with large-scale storage sites, more flexible methods for data layouts are needed. The basic Ceph architecture includes a meta-data server that is used to translate requests from a virtual storage interface to the physical disks [28]. In such systems, changes to meta-data in the server allows the administrator to migrate easily data blocks from batches of disks about to be removed to batches of new disks [29]. Ceph is used in conjunction with replication, though the basic architecture can easily be extended to work with reliability mechanisms based on erasure-resilient coding. One contribution of this paper is the extension of the architecture to simplify changes in the erasure-resilient codes.

Using only parity data obtained by exclusive-or operations in order to construct layouts that can tolerate more than a single failed device has been looked at since the late eighties. Gibson, Hellerstein and colleagues are to our knowledge the first ones to place data blocks in a two-dimensional matrix where each row and columns is augmented with blocks containing the exclusive-or of the row or column, respectively [9], [12]. Zhou *et al.* made the for us crucial observation that generalizations of this layout can be represented as a graph. They did not however evaluate the reliability of the resulting constructions [35].

## IX. Conclusions

We have proposed the introduction of a "virtual disklet layer" as a consistent abstraction of the underlying data storage to achieve better manageability of very large data centers and achieve energy savings. We proposed and evaluated a new reliable layout for this virtual disklet layer called RESAR. This layout places every disklet into two reliability groups with one parity disklet each. It has been designed to avoid small RAID writes (where we obtain the new parity from the delta of the new data, old data, and the old parity). We compared it with a layout organized as a collection of RAID Level 6 reliability stripes and showed the RESAR layout to be much more resilient. Of course, faulty disk array controller software has caused many instances of data loss in disk arrays and this type of error can destroy the reliability of the best layout. Choosing RESAR and Ceph for the disk layout only simplifies its implementation.

In the age of fast Galois field calculations [19], the fact that RESAR only uses exclusive-or operations does not necessarily speed up parity calculation, but it allows us to use cheaper and less power-hungry processors.

We have shown that introducing a virtual disklet layer simplifies administration, that a graph visualization simplifies argumentation, and that RESAR shows higher robustness and hence better reliability against disk failures than a layout based on RAID Level 6 stripes with the same parameters.

## Acknowledgments

This research was supported in part by National Science Foundation grants CNS-1528179, CCF-1219163 and IIP-1266400, by the Department of Energy under award number DE-FC02-10ER26017/DE-SC0005417, and by the industrial members of the Center for Research in Storage Systems.

## References

[1] G. A. Alvarez, W. A. Burkhard, L. J. Stockmeyer, and F. Cristian, "Declustered disk array architectures with optimal and near-optimal parallelism," *ACM SIGARCH Computer Architecture News*, vol. 26, no. 3, pp. 109–120, 1998.

[2] A. Amer, D. D. Long, J.-F. Paris, and T. Schwarz, "Increased reliability with SSPiRAL data layouts," in *IEEE International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems (MASCOTS)*. IEEE, 2008, pp. 1–10.

[3] A. Brinkmann, M. Heidebuer, F. M. auf der Heide, U. Rückert, K. Salzwedel, and M. Vodisek, "V: Drive-costs and benefits of an out-of-band storage virtualization system," in *Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2004, pp. 153–157.

[4] F. Chang, M. Ji, S.-T. Leung, J. MacCormick, S. Perl, and L. Zhang, "Myriad: Cost-effective disaster tolerance," in *USENIX Conference on File and Storage Technologies (FAST)*, 2002.

[5] L.-P. Chang, "On efficient wear leveling for large-scale flash-memory storage systems," in *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC)*. ACM, 2007, pp. 1126–1130.

[6] I. Corderi, T. Schwarz, A. Amer, D. D. Long, and J. F. Pâris, "Self-adjusting two-failure tolerant disk arrays," in *5th Petascale Data Storage Workshop (PDSW)*. IEEE, 2010, pp. 1–5.

[7] G. Dong, N. Xie, and T. Zhang, "On the use of soft-decision error-correction codes in NAND flash memory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 58, no. 2, pp. 429–439, 2011.

[8] V. E. Galinanes and P. Felber, "Helical entanglement codes: An efficient approach for designing robust distributed storage systems," in *Stabilization, Safety, and Security of Distributed Systems*. Springer, 2013, pp. 32–44.

[9] G. A. Gibson, L. Hellerstein, R. M. Karp, and D. A. Patterson, "Failure correction techniques for large disk arrays," in *3rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 1989, pp. 123–132.

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

[11] J. L. Hafner, "WEAVER codes: Highly fault tolerant erasure codes for storage systems." in *USENIX Conference on File and Storage Technologies (FAST)*, 2005.

[12] 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.

[13] M. Holland and G. A. Gibson, "Parity declustering for continuous operation in redundant disk arrays," in *5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1992.

[14] M. Holland, G. A. Gibson, and D. P. Siewiorek, "Architectures and algorithms for on-line failure recovery in redundant disk arrays," *Distributed and Parallel Databases*, vol. 2, no. 3, pp. 295–335, 1994.

[15] G. F. Hughes, J. F. Murray, K. Kreutz-Delgado, and C. Elkan, "Improved disk-drive failure warnings," *IEEE Transactions on Reliability*, vol. 51, no. 3, pp. 350–357, 2002.

[16] K. Jian, Z. Xu-Dong, N. Wen-wu, Z. Jun-wei, H. Xiao-ming, Z. Jian-gang, and X. Lu, "A performance isolation algorithm for shared virtualization storage system," in *International Conference on Networking, Architecture, and Storage (NAS)*. IEEE, 2009, pp. 35–42.

[17] H. Liu, S. Liu, X. Meng, C. Yang, and Y. Zhang, "LBVS: A load balancing strategy for virtual storage," in *International Conference on Service Sciences (ICSS)*. IEEE, 2010, pp. 257–262.

[18] C. R. Lumb, A. Merchant, and G. A. Alvarez, "Façade: Virtual storage devices with performance guarantees." in *USENIX Conference on File and Storage Technologies (FAST)*, 2003.

[19] J. S. Plank, K. M. Greenan, and E. L. Miller, "Screaming fast Galois field arithmetic using Intel SIMD instructions," in *USENIX Conference on File and Storage Technologies (FAST)*, 2013.

[20] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *Second International Conference on Computer and Network Technology*, ser. ICCNT '10. IEEE, 2010, pp. 222–226.

[21] E. J. Schwabe and I. M. Sutherland, "Improved parity-declustered layouts for disk arrays," *Journal of Computer and System Sciences*, vol. 53, no. 3, pp. 328–343, 1996.

[22] ——, "Efficient data mappings for parity-declustered data layouts," *Theoretical Computer Science*, vol. 325, no. 3, pp. 391–407, 2004.

[23] T. Schwarz, D. D. Long, and J. F. Pâris, "Reliability of disk arrays with double parity," in *19th Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE, 2013, pp. 108–117.

[24] T. Schwarz and W. A. Burkhard, "Almost complete address translation (acats) disk array declustering," in *8th Symposium on Parallel and Distributed Processing*. IEEE, 1996, pp. 324–331.

[25] T. J. Schwarz, J. Steinberg, and W. A. Burkhard, "Permutation development data layout (pddl)," in *5th International Symposium on High-Performance Computer Architecture*. IEEE, 1999, pp. 214–217.

[26] T. J. Schwarz, Q. Xin, E. L. Miller, D. D. Long, A. Hospodor, and S. Ng, "Disk scrubbing in large archival storage systems," in *International Symposium on Modeling, Analysis and Simulation of Computers and Telecommunication Systems (MASCOTS)*. IEEE, 2004, pp. 409–418.

[27] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers," in *ACM/IEEE Conference on Supercomputing*, 2008.

[28] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *7th symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2006, pp. 307–320.

[29] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "Crush: Controlled, scalable, decentralized placement of replicated data," in *ACM/IEEE Conference on Supercomputing*, 2006.

[30] E. B. Wilson, "Probable inference, the law of succession, and statistical inference," *Journal of the American Statistical Association*, vol. 22, no. 158, pp. 209–212, 1927.

[31] Q. Xin, E. L. Miller, T. Schwarz, D. D. Long, S. A. Brandt, and W. Litwin, "Reliability mechanisms for very large storage systems," in *Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2003, pp. 146–156.

[32] L. Xu, V. Bohossian, J. Bruck, and D. G. Wagner, "Low-density MDS codes and factors of complete graphs," *IEEE Transactions on Information Theory*, vol. 45, no. 6, pp. 1817–1826, 1999.

[33] W. Zeng, Y. Zhao, K. Ou, and W. Song, "Research on cloud storage architecture and key technologies," in *2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*. ACM, 2009, pp. 1044–1048.

[34] J. Zhang, A. Sivasubramaniam, Q. Wang, A. Riska, and E. Riedel, "Storage performance virtualization via throughput and latency control," *ACM Transactions on Storage*, vol. 2, no. 3, pp. 283–308, 2006.

[35] J. Zhou, G. Wang, X. Liu, and J. Liu, "The study of graph decompositions and placement of parity and data to tolerate two failures in disk arrays: Conditions and existance," *Chinese Journal of Computers*, vol. 26, no. 10, pp. 1379–1386, 2003.