Self-Adjusting Two-Failure Tolerant Disk Arrays

Ignacio Corderí, Thomas Schwarz,S.J.
Informática y Ciencias de la Computación
Universidad Católica del Uruguay
Montevideo, Uruguay
ignacio@corderi.com tschwarz@ucu.edu.uy

Ahmed Amer
Computer Engineering
Santa Clara University
Santa Clara, CA
a.amer@acm.org

Darrell D.E. Long
Computer Science
University of California
Santa Cruz, CA
darrell@cs.ucsc.edu

Jehan-François Pâris
Computer Science
University of Houston
Houston, TX
paris@cs.uh.edu

I. Introduction

While flash and Storage Class Memory (SCM) technologies stand to replace magnetic disk technology as the mainstay for high end applications, the sheer amount of data to be stored, the attractive cost-to-capacity ratios of disks, and the high streaming throughput in comparison with not only tape but also with high-end flash and SCM, give magnetic disk technology a continuing and important role in the storage hierarchy. This is true whether disks are relegated to tertiary storage roles or remain as the secondary storage technology behind flash/SCM-based caches. A practical disk-based storage system at petabyte scale is both dynamic and heterogeneous, as the number of devices it would require means that new disks with better performance, reliability, and capacity will continuously enter the system as old disks leave due to failure, age, or technical obsolescence.

Data stored in a peta-scale system needs to be protected, but its size will make disk failure a daily occurence. Observed data on the life expectancy of disks [1], [2] and the occurrence of latent disk sector errors [3] suggest that tolerance of at least two failures is necessary, possibly in conjunction with disk scrubbing or intra-disk redundancy [4].

While replication offers operational advantages, the storage overhead with its associated costs in hardware and energy is too large. Many two-failure resilient systems have been proposed in the past [5], [6], [7], [8]. We propose an old, simple scheme in which every piece of client data is part of two different reliability stripes encompassing data disks and one additional parity each. Managing this simple layout over the lifetime of an evolving system is difficult. Our contribution is a graph-based representation that transforms layout decisions into the construction of (almost) regular graphs and coloring their edges and vertices with many colors. For this, we can use simple, greedy and heuristic graph algorithms.

II. Graph Representation

We store client data in disklets, virtual disks of fixed size stored contiguously in the physical disks of the system. Using disklets allows us to deal with the dynamism and heterogeneity of the storage system, which at any time could contain disks of varying generations and capacities. Disklets can be moved transparently to the user between physical devices. The size of the disklets offers a trade-off. Fewer, larger disklets are easier to administer. More, smaller disklets fit better into the disks of varied sizes. Intuitively, we propose using disklets of about 100GB, which would mean that current disks would hold about ten disklets each, with future generations introduced into the storage system holding more. At this rate, no more than 100GB (less than 10% of current high capacity disks) could potentially go to waste. Space reasons prevent us from discussing how even such apparently wasted storage space could yet be used.

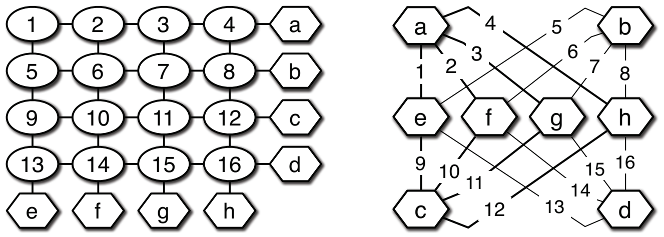

Fig. 1: A small two-failure resilient array and its design-theoretical dual (right).

A. Disklets, Reliability Strips, Configurations and a Dual

We distinguish between disklets that store client data and disklets that store parity data (*i.e.*, used only to recover from disk failures). We place each data disklet into groups of $n-1$ disklets to which we add a single parity disklet. We call the resulting ensemble of n disklets a reliability stripe. Data on a single lost disklet in a reliability group can be recovered by reading from all the other members of the reliability group. To withstand simultaneous double failure, we place each data disklet in two different reliability stripes. Of course, as disklets on a single failed disk are likely to fail at the same time, we will have to deal with common failure causes, but we will come to that. A parity disklet belongs to only one reliability stripe. A very simple example of such an arrangement is given in Fig. 1, left. The ovals with numerical values represent data disklets, while the hexagons with letters represent parity disklets. The arrangement of disklets in rows and columns represent assignment to reliability stripes. For example, data disklets 13, 14, 15, and 16, together with parity disklet d, form a reliability stripe.

To use mathematical Design Theory (from finite mathematics), we call each data disklet an *element* and the set of data disklets in a reliability group a *block*. Each parity disklet corresponds to exactly one block, namely the parity is calculated for the group of disklets in the block. A two-(disklet)-failure tolerant layout consists of elements organized

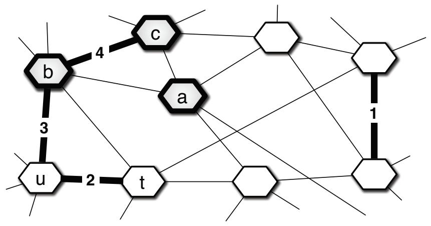

Fig. 2: Failure pattern.

into blocks such that:
(1) Each element is in exactly two blocks.
(2) Each block contains exactly n elements.
(3) Two different elements are in at most one block.
These properties define a configuration [9]. We can obtain additional insight by passing to the *dual* design, where the blocks of the dual correspond to the elements of the primary design, the elements of the dual to the blocks of the primary, and where the "is an element of" relationship between elements and blocks is reversed. In the dual, the elements are reliability stripes (represented by the parity disklets), the blocks are data disklets, and a dual element (parity disklet) belongs to a block (data disklet) if the data disklet is part of the reliability stripe that stores its parity in the parity disklet. As each data disklet is in exactly two reliability stripes, the dual is a graph. Since each parity disklet stores the parity of n data disklets, the dual is n-regular (*i.e.*, the node degree of the graph is n). Fig. 1 right gives the dual of the design on the left. For example, the reliability stripe consisting of data disklets $\{3,7,11,15\}$ stores its parity on disklet g. Data disklet 7 is also in stripe $\{5,6,7,8\}$ with parity in b, therefore, in the dual, 7 is an edge between b and g.

Conversely, given an n-regular graph, we obtain the dual disklet array layout by adopting the following rules:
(1) Each vertex corresponds to a parity disklet.
(2) Each edge corresponds to a data disklet.
(3) The parity disklet contains the XOR of all data on disklets adjoining it.
Graph theory knows of many families of n-regular graphs.

Disklets need to be housed in real disks. We represent this by coloring the edges and vertices of the graph with the disk number. Integration of new disks, removal of obsolete disks, and data recovery after failure force us to change the disklet layout. These changes are represented by manipulations in the graph. If we introduce a new parity disklet, we add a vertex to the graph. If we introduce a new data disklet, we add an isolated edge to the graph. To attach this edge, corresponding to disklet i with data D_i to a parity disklet a with content P_a we need to calculate $P_a := P_a \oplus D_i$. To detach disklet i from parity disklet a, we also calculate $P_a := P_a \oplus D_i$. Removing parity disklets does not change data on other disklets.

B. Failure Tolerance Representation

Arguments about failure tolerance are much easier in the graph than in the original primary design, as was previously observed [10]. Disk and sector failure induce a *failure pattern* in the graph, *i.e.*, the set of the now unavailable disklets. Fig. 2 gives an example where three failed parity disklets (vertices) and four failed data disklets (edges) are marked. Often, we can *reconstruct* unavailable disklets, which means calculating the lost data and placing them on a replacement disklet stored elsewhere. The graph representation only indicates this process by coloring with a different disk color and unmarking the disklet.

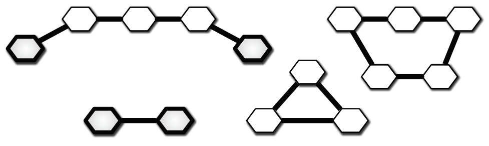
Fig. 3: Minimal irreducible failure patterns.

Data on a lost parity disklet is recovered by accessing all data disklets in the same reliability stripe and recalculating the parity. Thus, we can unmark a failed vertex if none of the adjacent edges is marked. Data on a data disklet can be recovered from all other disklets in the same reliability stripe. Thus, we can unmark a failed edge if one of its adjacent vertices and all other edges adjacent to that vertex are not marked. Edge 1 in Fig. 2 fullfills this condition with regards to both vertices, but edge 2 solely at vertex t. Vertex a can also be immediately recovered. Data recovery often has to be iterative. We can immediately recover a, 1, and 2, but only after recovering 2 can we recover 3. After this, we are stuck, since we cannot make progress for the remaining b, c, and 4. Thus, Fig. 2 represents a situation where we have lost data. We call this process of unmarking recoverable items *failure pattern reduction.*

A simple linear algorithm for failure pattern reduction anotates marked edges and vertices with positive numbers. A marked vertex gets the number of marked, adjacent edges and a colored edge (x,y) receives two numbers, f_x and f_y. If x is marked, then f_x is the number of adjacent marked edges, otherwise it is that number minus 1. Analogously for f_y. For example, edge 3 has $f_u = 1$ and $f_b = 2$. We can unmark if one of these numbers is zero. In this case, adjacent edges and vertices change their number.

Not all failure patterns are reducible. These irreducible failure patterns describe instances of data loss. For an edge to be part of an irreducible failure pattern, either the end-vertices also failed, or at least one of the adjoining edges has also failed, or both. Therefore, minimal irreducible failure patterns are either a chain, Fig. 3 left, or a cycle, Fig. 3 right. The chain is a walk starting and ending at a failed vertex connected with failed edges in between. The cycle is an edge cycle in the sense of graph theory. The smallest minimal failure patterns are the *bar-bell* (Fig. 3, lower left), and the triangle (Fig. 3, lower right).

C. Good Disklet Layouts

The number n of data disklets per reliability stripe determines the parity storage overhead and the recovery load. The storage overhead (amount of parity data divided by the amount of client data) is $2/n$ as each data disklet is part of two reliability stripes. As we assume a large storage system

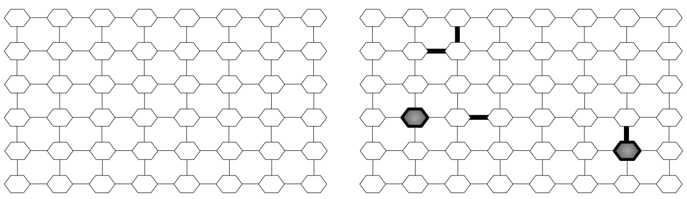

Fig. 4: Grid graph without and with failure pattern.

with Flash / SCM components, we can assume that the cost of maintaining parity is low. Recovering data stored on a lost disklet involves reading n and writing one disklet. The choice of n offers therefore a trade-off between parity overhead and reconstruction load. Smaller reconstruction load can lead to faster recovery and therefore to higher reliability. We can gain flexibility by allowing the parameter n to vary slightly between reliability stripes. In the graph, this corresponds to an "almost regular" graph where the number of edges connected to a vertex is not constant, but varies slightly.

Since several disklets are located on the same disk, a disk failure results in the loss of a number of disklets. When we use a good disklet-to-disk assignment (to be discussed shortly), a double disk failure will not cause data to be lost. However, an adversary can always choose three disks to fail so that data loss occurs. We want to minimize the possibility that three disk failures (or more) result in the occurrence of either the barbell or the triangle. We cannot prevent the former, but we can avoid triangles through the shape of the graph.

To create a good initial disklet layout, we therefore choose an n-regular graph without triangles. Among the many possibilities, we can select subgraphs of a *grid graph*. The grid graph has for vertices the points $\mathbb{N}^r$ with integer coefficients in the r-dimensional Euclidean space. The edges are defined to be the lines of length 1 between these points. The edges therefore only connect vertices that differ in only one coordinate, and in that coordinate by one. The results is an r-dimensional grid.

After choosing the graph, we select a finite region, for example defined by a square box consisting of vertices with coordinates less than a given l. We then assign disklets to the vertices in the box, these will become the parity disklets. Afterwards, we assign data disklets to the edges. The subgraph is not regular. An interior vertex has degree $2r$, but one of the $2r$ corner vertices has only degree r. To make the graph regular, we connect vertices on the boundary to its diametrically opposed counter-vertex. In fact, we can avoid this last operation because it complicates growing the graph (necessary if the disk array increases in size). The vertices with fewer edges correspond to reliability stripes with fewer data disklets and cause higher parity overhead, but since most of the vertices are interior (for a disk array of substantial size), the overall loss is easy to tolerate. In general, we do not need to assign all disklets in an array at the same time. In this case, we can postpone the decision of having a regular graph or an almost regular graph until we are either about to run out of space (in which case we assign data disklets to the edges between diametrically opposite vertices) or about

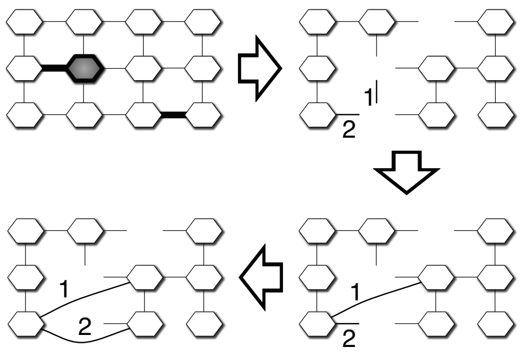

Fig. 5: Beginning of restructuring after failure.

to incorporate new disks (in which case we want to grow the array). Finally, the subgraph does not have to be a square box, but can be rectangular.

D. Disklet to Disk Assignment

A disklet layout is defined by an (almost) n-regular graph. We represent the assignment of disklets to disks by coloring the element (vertex or edge) with a color representing the disk. Not every coloring will do, as otherwise a single disk failure might lead to data loss. We color elements with the same color (*i.e.*, collocate disklets on the same disk) if they are apart from each other in the graph. We now discuss our notion of distance.

We use the notion of *walks* in graphs. A walk is a sequence of alternating edges and vertices that are adjacent to each other. Graph theory defines the length of a walk to be the number of edges contained in it, but we use here a different measure, namely the number of elements in a walk minus one. In Fig. 2, the sequence t, 2, u, 3, b, 4, c is therefore a walk of length six. We define the *walking distance* between two elements as the length of a minimal walk (in our sense) connecting them. Elements in an irreducible failure pattern have to be at walking distance from each other. Consequently, if elements colored with the same disk are at least at walking distance two, then no two disk failures can lead to data loss. Coloring the graph subject to this restriction is fairly simple because of the large number of colors (disks).

E. Restructuring after Failure

Assume that a number of disks have failed simultaneously and that recovery operations or scrubbing has led to the discovery of a number of disklets with latent sector errors. As a result, our graph now has some failed elements. Fig. 4 gives an example. To the left, we show the grid graph used. The graph is two-dimensional and the interior vertices have edge degree of only 4 corresponding to reliability stripes of size $4+1$. In reality, we would use a grid of higher dimension. To the right, we show the effect of a disk failure combined with latent sector failures.

To recover from failure, we reconstruct data and have to place them in disklets on disks. If we have empty disklets, then the reconstructed data can be placed on some of these disklets. However, this placement recolors the graph and we have to take care that our rule against a minimal walking distance between two elements with the same color is preserved. If we cannot avoid a violation by assigning reconstructed disklets to disks, then we pick two disklets and interchange their location (This is not a small operation as disklets are tens of GB in size).

An additional problem arises if there are no free disklets to store reconstructed data. While this occurs rarely (most storage systems are not left to reach full capacity), we need to handle this case. We do so by reorganizing as previously proposed [11]. The idea is to restructure by enlarging reliability stripes and thus freeing parity disklets for use as data disklets. In the example of Fig. 5, we have three failed elements, but have not suffered data loss. We need two spots to store the data from the two failed data disklets. We pick randomly two parity disklets and use them for this purpose. In our graph representation, we now have several dangling edges, including one that is not attached to any vertex. This corresponds to a data disklet that does not contribute to any parity and is therefore not protected against failure. We now *attach* dangling edges to vertices. This corresponds to XORing the contents of these data disklets to the parity disklets. Fig. 5 lower left shows the result after reattaching two edges. There still remain seven dangling edges to be reattached. The resulting graph is not pretty, since it is now far from being regular and because we cannot avoid triangles. However, as long as we preserve the graph property, we are still protected against double disk failure. Attaching dangling edges might also lead to violation of the coloring rule that two elements of the same color not be placed close together. In a small graph like this, it is difficult to enforce this rule successfully, but in a graph representing a petabyte-scale storage system, the sheer number of disks means that using a simple greedy algorithm would be feasible.

F. Incorporating New Disks

New disks usually enter the system in large numbers, *e.g.*, one or more racks full of disks needing to be incorporated. The following strategies are possible. First, we can just configure the new disks independently of the old ones. The resulting graph then has at least two components, one of which is colored only for the new disks. If a new disk fails, recovery workload will be distributed over only the new disks. The second strategy distributes the recovery workload over all disks and incidentally addresses the problem of disk infant mortality. However, the cost is a major reshuffle of data moving many disklets from old disks to new ones.

The first strategy just adds a new component to the graph. The second is represented by adding more edges and vertices and recoloring some edges and vertices to represent the copying of disklets from old disks to disklets on new disks. To adjust the number of disklets, we increase the box that defined the grid graph. Depending on how we dealt with lack of regularity, this might imply detaching edges on the boundary of the box. Detaching an edge means removing a data disklet from a parity stripe. This is done by XORing the contents of the data disklet and the contents of the parity disklet. Then, if we move a disklet from the old part of the array to a disk in

TABLE I: Data loss rate and data loss probability after failure of a rack and simultaneous, additional x disk failures (for 20 racks with 50 or 1000 disks, and 50 racks with 400 disks).

x	20×50		20×1000		50×400 disks	
	DLP	DLR	DLP	DLR	DLP	DLR
1	0.02%	0.014%	10.76%	0.0005%	0.000%	0.0000%
2	0.32%	0.014%	20.32%	0.0005%	0.002%	0.0001%
3	1.00%	0.013%	29.02%	0.0006%	0.020%	0.0004%
4	2.08%	0.014%	36.82%	0.0006%	0.034%	0.0006%
5	3.67%	0.014%	43.55%	0.0006%	0.044%	0.0006%
10	15.95%	0.017%	68.51%	0.0008%	0.282%	0.0008%
20	58.75%	0.025%	90.49%	0.0012%	1.06%	0.0007%

the new part, we give it a new color, specifically the name of the disk. Afterwards, we color the new edges and vertices in the graph.

G. Removing Obsolete Disks

Disks leave the system because of failure (in which case we can either recolor the corresponding disklets to disks that remain in the system if the disks have space left for the disklets, otherwise we have to reconfigure) or because they have become obsolete. Data on obsolete disks would be moved elsewhere in the storage system, possibly on replacement disks. In the graph, this is once again expressed as recoloring.

H. Reconfiguring for Energy Savings and Load Balancing

The load of a disklet depends on the data it contains. By swapping disklets between disks, we can achieve load balancing or congregate low demand disklets on disks that can be turned off, without changing the resilience of the disk array. If we can identify disklets with read-only data, we can also use swapping and rearranging of reliability stripes to have parity disklets that have no load. Such disklets can also be placed on disks to be powered off.

III. System Design

Our representation / management approach applies to all large storage systems and presupposes only the capability to maintain a large graph and run greedy algorithms on that graph. This is certainly the case if we have a metadata server, which would also be needed to implement search functionality over the large amount of data and to manage the relationship between a potential first level storage system using solid state disks and our disk-based system.

To the clients, our storage system gives the abstraction of small, but highly reliable virtual disks, formed of the disklets. While any such abstraction can result in less complete use of the available storage capacity, it does facilitate energy savings by turning off disks needed only by dormant clients, and it should also increase spatial locality. The scheme is similar to the extents used in database systems to administer large amounts of storage space.

IV. Preliminary Experimental Results

We started implementing algorithms based on our graph representation. We implemented the initial disklet layout for a system comprised of homogeneous racks containing disks with 10 disklets each. We used a layout where 8 data disklets

TABLE II: Probability of data loss in an array with N disks and x failures.

N	$x=3$	$x=4$	$x=5$	$x=6$	$x=7$	$x=8$	$x=9$	$x=10$
200	0.362%	1.154%	2.266%	4.248%	7.020%	10.562%	15.518%	21.458%
500	0.048%	0.130%	0.346%	0.676%	0.936%	1.480%	2.264%	3.304%
1000	0.020%	0.034%	0.092%	0.158%	0.260%	0.468%	0.616%	0.768%
1500	0.008%	0.022%	0.044%	0.082%	0.134%	0.160%	0.298%	0.354%
2000	0.004%	0.022%	0.022%	0.022%	0.044%	0.082%	0.134%	0.160%
5000	0.002%	0.002%	0.004%	0.014%	0.008%	0.010%	0.012%	0.050%
10000	0.000%	0.000%	0.000%	0.000%	0.000%	0.004%	0.004%	0.010%
15000	0.000%	0.000%	0.002%	0.002%	0.000%	0.002%	0.000%	0.004%
20000	0.000%	0.000%	0.000%	0.002%	0.002%	0.000%	0.000%	0.002%

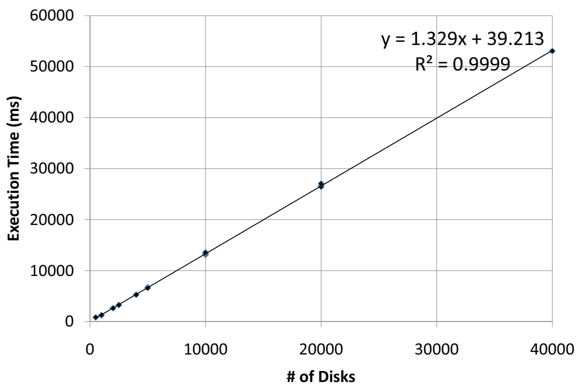

Fig. 6: Execution times for initial graph layout.

form a reliability stripe with one additional parity disklet. The execution times for this initial data layout depended linearly on the number of disks (1.329 ms per disk), but not on the number of racks. Accordingly, this (one-time) task can be done in less than a minute for an array with 20000 disks (Figure 6).

Our layouts do not suffer data loss from any double failure, whether rack, disk, or sector, with the exception of two rack failures, but any combination of three failures can lead to data loss. We have to measure resilience carefully. An experiment repeated multiple times will have a bad outcome with probability arbitrarily close to one, if we only repeat it enough times. Similarly, if we increase the number of disklets per disk, then any combination of more than two disk failures will, with high probability, result in data loss. We have to measure the *robustness* of a layout not only by the probability of data loss given a certain combination of failures, but also by the expected rate of data loss.

Our first experiment simulated the robustness of the disk array right after the failure of a whole rack. The results, in Table I give the Data Loss Rate (DLR) and Data Loss Probability (DLP) using 10 disklets per disk. The DLR is very good, given that a rack constitutes 1/20 or 1/50 of all disks in the array. The DLP is high with few racks. In our second experiment (Table II) we observed good robustness of our layout using again with 10 disklets per disk. We do not give DLR because we usually only lose one data disklet if we lose data at all. At worst, we lose 1.68 data disklets on average after 10 disk failures out of a total of only 200. When we varied the number of disklets, we observed a slightly lower DLR coupled with a much increased DLP.

V. Conclusions and Future Work

We have presented a representation for a storage system with two failure tolerance based on flat XOR codes. We argue that this representation allows us to implement fast algorithm for the layout of very large, evolving disk arrays.

Much needs to be done. Fast, but efficient algorithms for major changes in the disk array such as rack failure or insertion of new disks still need to be implemented and tested. Our goal is usually not to find an optimal layout (in a sense to be defined precisely), but one that is close to optimal. To assert that our algorithms perform at this level involves a more mathematical analysis of the consequences of failures in such an array to derive bounds on the robustness of optimal layouts, a task we have barely started. Nevertheless, the results we have indicate that the algorithms are quite effective and certainly fast and easy to implement. This presents definite progress over the true optimization (including looking for proven optimal designs) that can be done only for special, small cases and supports our pragmatic attitude.

References

[1] E. Pinheiro, W. Weber, and L. Barroso, “Failure trends in a large disk drive population,” in *Proc. 5th USENIX FAST Conf.*, 2007.
[2] B. Schroeder and G. Gibson, “Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?” in *Proc. 5th USENIX FAST Conf.*, 2007.
[3] L. Bairavasundaram, G. Goodson, S. Pasupathy, and J. Schindler, “An analysis of latent sector errors in disk drives,” in *ACM SIGMETRICS*, 2007.
[4] I. Iliadis, R. Haas, X. Hu, and E. Eleftheriou, “Disk scrubbing versus intra-disk redundancy for high-reliability raid storage systems,” in *ACM SIGMETRICS*, 2008.
[5] M. Blaum, J. Brady, J. Bruck, and J. Menon, “EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures,” *IEEE Transactions on Computers*, pp. 192–202, 1995.
[6] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, “Row-diagonal parity for double disk failure correction,” in *Proc. 3rd USENIX FAST Conf.*, 2004.
[7] K. Greenan, X. Li, and J. Wylie, “Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs,” in *Proc. IEEE MSST*, 2010.
[8] L. Hellerstein, G. Gibson, R. Karp, R. Katz, and D. Patterson, “Coding techniques for handling failures in large disk arrays,” *Algorithmica*, vol. 12, no. 2, pp. 182–208, 1994.
[9] H. Gropp, “Configurations,” in *The CRC Handbook of Combinatorial Designs*, C. Cobourn and J. Dinitz, Eds. CRC Press, 1996.
[10] Z. Jie, W. Gang, L. Xiaogugang, and L. Jing, “The study of graph decompositions and placement of parity and data to tolerate two failures in disk arrays: Conditions and existence,” *Chinese Journal of Computers*, vol. 26, no. 10, pp. 1379–1386, 2003.
[11] J.-F. Pâris, T. Schwarz, and D. Long, “Self-Adaptive Two-Dimensional RAID Arrays,” in *Proc. IEEE IPCCC*, 2007.