String Layout in Redundant Disk Arrays

Lee Newberg* and David Wolfe
University of California, Berkeley

Abstract

It has become economically advantageous to use arrays of inexpensive disks in place of fewer large disks. Some care must be taken to maintain high reliability in these new systems; adding extra disks to store redundant information achieves a system which is able to withstand some disk failures. The problem is exacerbated, however, when a controller or power line fails, and a string of disks go down simultaneously. We discuss how the strings should be laid out to maximize reliability of the system.

1 Introduction

Improvements of CPU performance have far exceeded the speedups achieved in I/O. CPU's can handle much higher bandwidths than can be provided by a small number of disks. [PGK] suggest using a *redundant array of inexpensive disks* or RAID to boost I/O bandwidth and reduce disk latency times. Small disks can spin faster, providing faster seek times and data transfers. An array of disks operating in parallel increases bandwidth when accesses are made to separate disks.

Unless we take special care, these advantages are at the expense of reliability. The likelihood of a single disk failure does not change significantly with the size of a disk. Thus if the number of disks is increased tenfold, the likelihood that a disk will be out also increases tenfold. RAID uses extra disks to store redundant information, so that data lost after disk failures is recoverable.

The implementation suggested in [GHKKP] is to use a square array of data disks, with an additional check disk in every row and column: A row (or column) check disk keeps the parity of all the data disks in its row (or column). Now each update to the disk array consists of updating three disks - a data disk and two check disks. [GHKKP] argue that the loss of any two disks is still recoverable, and furthermore this arrangement is nearly

*Supported by a National Science Foundation Graduate Fellowship

optimal among those that update only three disks. This implies high reliability if disk failures are uncorrelated.

However, some disks failures are correlated. Typically, several disks are connected to the same power supply and are controlled by the same controller. Several disks on such a *string* can all fail simultaneously. We discuss how these strings should be laid out to maintain high reliability of the system.

2 Model and Definitions

2.1 2d-Parity*

We consider two layouts of disks. The first is called *2d-parity*. Data is stored, unaltered, in an $m \times m$ square array of *data disks*. In addition there are $2m$ *check disks* which are used to store parity information. Each row and each column has its own check disk. A row check disk is placed at the right end of its row and a column check disk is placed at the bottom of its column. See Figure 1.

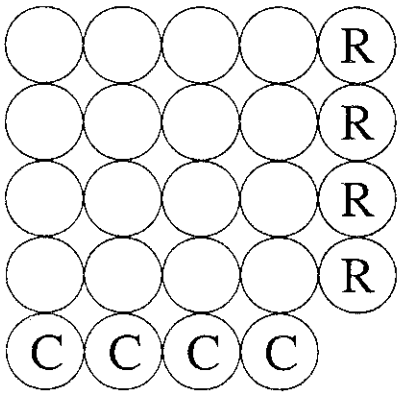

Figure 1: *The 2d-parity layout of disks*

The check disks store parity information for their row or column. Each bit on the check disk for a given row (or column) is computed from the corresponding bit from each disk in that row (or column). The bit on the check disk is the parity (i.e. exclusive-or) of the corresponding bits on the data disks.

Disk reads from this array are straight forward. The required data is available unencoded from a single disk. Writes are a little more complicated. Parity information must be updated as well as the data information. Each write will therefore involve three disks — one data disk, one row check disk, and one column check disk.

*A glossary of terms is located in Appendix B. Also, theorems and claims referenced in the paper are presented in Appendix A.

The check disks enable the system to retrieve data even when some of the data disks have failed. If a single disk in a row has failed its contents can be computed from the remaining data disks on the row and the row's check disk. The exclusive-or of the bytes on these disks gives the bytes desired.

Sometimes when a collection of disks fails data can be reconstructed and sometimes it cannot. [GHKKP] give an algorithm that determines how to retrieve data if it is possible. They show that if any two disks fail all information can be retrieved. There are some combinations of three disks failing that will result in loss of data but they are not common. The only triplets that cause problems are those consisting of a data disk, its row check disk, and its column check disk.

2.2 Complete 2d-Parity

By adding more redundancy a system can guarantee the availability of data when *any* three disks have failed. One such system involves adding one more check disk which stores the parity of *all* the data disks. This *total parity check disk* is placed in the lower righthand corner of the array, below the row check disks and to the right of the column check disks. We call this model *Complete 2d-parity*. See Figure 2.

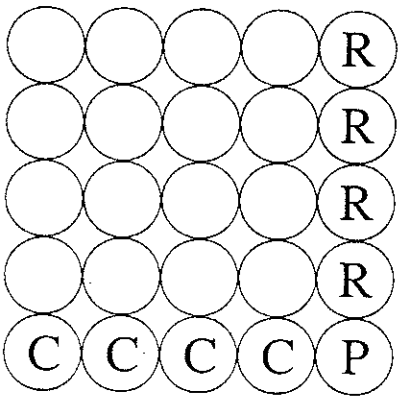

Figure 2: *The* complete 2d-parity *layout of disks*

Complete 2d-parity is reliable enough to survive the failure of any three disks. A combination of four disks will result in the loss of data only when the four disks comprise the four corners of a square. This is true of any square whether the four disks are all data disks or some are check disks. Figure 3 shows a bad square.

The blurring of the distinction between data disks and check disks is what makes complete 2d-parity interesting from a theoretical standpoint. All the disks have exactly the same recovery properties regardless of whether they are data disks or check disks. This strong symmetry property makes it possible to prove many theorems about complete 2d-parity. These, in turn, can be used to describe the normal 2d-parity array.

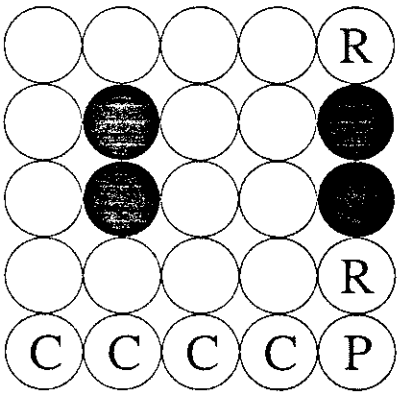

Figure 3: *A* bad 2-square.

Unfortunately, although complete 2d-parity is useful for theoretical purposes, it is not so useful in application because a write involves four disks. As DRAM prices drop it may become feasible to place the total parity information in memory. In this case data writes are reduced to just three disks which is as efficient as normal 2d-parity.

2.3 Data Planes

The main advantage of an array of disks is the increase in I/O bandwidth that it provides. Reads or Writes to different disks can occur simultaneously. An array of $m \times m$ data disks can support up to m^2 reads at a time.

Three disks are involved in writes to a 2d-parity array because a data disk and two check disks must be written. Writes to separate data disks may occur simultaneously only if they are not in the same row and not in the same column. If two data disks are in the same row (or column) they share the same row (or column) check disk and therefore one write must wait for the other.

To prevent this write bottleneck different *data planes* are introduced. Conceptually the disk arrangement is similar to the arrangement for complete 2d-parity. An extra row and column is added to the $m \times m$ array of *data* disks giving an $n \times n$ array of disks where $n = m + 1$. Each disk is partitioned into sections in an identical fashion. A section of one disk along with the corresponding section from each of the other disks form a data plane.

Each data plane can use a different set of disks for check disks. This has the advantage that two writes to disks on the same row will not contend for the same check disk unless both writes are to the same data plane. Each data plane has exactly one *pivot disk* which uniquely determines the location of the check disks on that plane. The row check disks are those disks in the same column as the pivot disk. The column check disks are those disks are in the same row as the pivot disk. The pivot disk itself is unused in the data plane. The remaining disks are for storing unencoded data. See Figure 4.

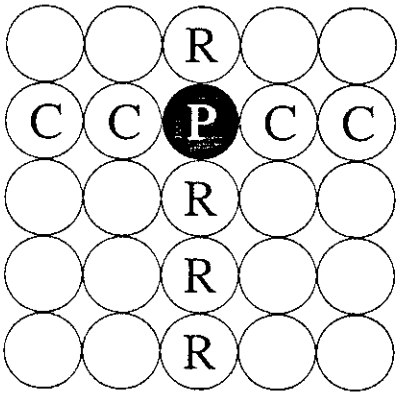

Figure 4: *The location of the check disks is uniquely determined by the location of the pivot disk.*

The 2d-parity arrangement we present later in the paper will have pivot disks along the main diagonal. Figure 5 shows where the pivot disks will be placed.

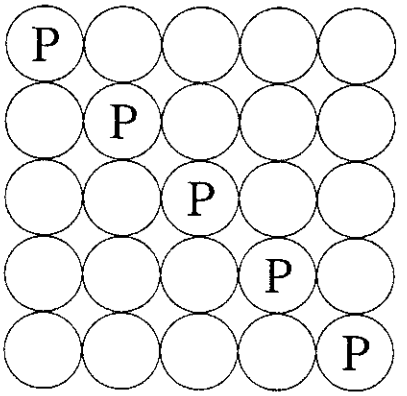

Figure 5: *Pivot disks are used to distribute parity information among all the disks in the array. Each row and column should have a pivot disk so that every disk (which is not a pivot disk) has the chance to be a check disk. The placement of pivot disks along the main diagonal of the array achieves this goal.*

3 Bad Squares

Both the design and the evaluation of a reliable string layout require an understanding of what combinations of disk outages are irrecoverable. For this we introduce the concept of a *bad square*.

A *bad square* or *bad cycle* is a minimal collection of disks whose failure will result in the loss of data in the 2d-parity arrangement. In this sense, minimal means that if any of the disks is magically repaired then it becomes possible to reconstruct the data. Minimality also implies that every collection of disks that causes a loss of data must contain a bad square.

In Theorem 1 we show that bad squares can only take on a very specific form. A *bad k-square* is a bad square consisting of $2k$ disks. Each of k columns and each of k rows has exactly two bad disks. It is possible to walk from disk to disk visiting each disk once in a cycle. Starting with one disk there is another disk in its row, which has another disk in its column, which in turn has another disk in its row, etc. Eventually the first disk is revisited as the walk comes from the disk in its column.

An example of the simplest bad square is the bad 2-square shown in Figure 3. In this case the walk is trivial. Starting with the upper left disk, the disks are visited in clockwise order. More complicated examples are shown in Figure 6.

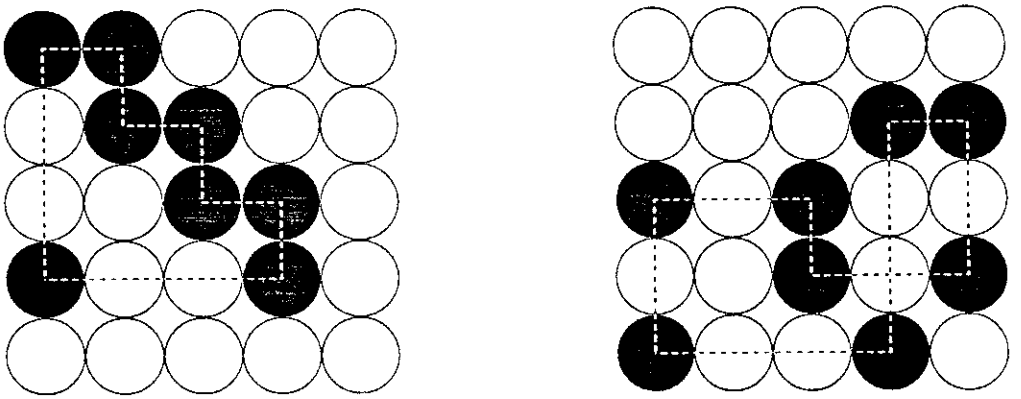

Figure 6: Bad k-squares for $k = 4$. On the left is a bad square, and on the right is the same bad square after a permutation on the rows and columns.

We now have a handle on what patterns of disk failures cause loss of data. If a subset of the disks that have failed contain a bad square then there is data loss; otherwise, there isn't. This will help with the design of string layout, since we know what patterns to avoid, and with calculating the mean time to failure of a RAID array.

4 String Layouts

We now consider the string and pivot disk layouts for a 2d-parity arrangement. We would like a layout which

1. Withstands failures: Naturally, we want to minimize the chance of data loss.

2. Uses few strings: Since controllers and power supplies can be costly, we want to use as few as possible. Furthermore, if we use too few strings (so that too many disks are on each string), we can divide the strings, and still maintain a good layout.

3. Distributes traffic: Check disks are a bottleneck for writes: Every write to a disk in a row writes to the same row check disk. So we want to distribute the parity information evenly.

First we discuss the limits on the reliability any disk layout can attain, and then we propose a layout that meets these limits.

4.1 Theoretical limits

Since we would like to distribute the parity, it is natural to make every disk in the disk array be a pivot for some plane – then every disk's role is totally symmetric. If disk writes are evenly distributed then so is the traffic. Unfortunately, as we show in Claim 1, no matter what the string layout, there always exists a pair of strings whose failure leads to data loss. We must, therefore, abandon the possibility of making every disk a pivot disk because of reliability concerns.

However, our objective is merely to distribute the check disks (and therefore, the traffic), so it is sufficient to place the pivot disks in such a way that there is one in each row and column. This way most disks have the opportunity to be a check disk for some plane. The placement of pivot disks along the main diagonal (See Figure 5) leads to an optimal layout.

Claim 2 shows that even in complete 2d-parity, there are always three strings whose loss is fatal. So our goal is to arrange the strings in 2d-parity so as to withstand all possible failures of 2 strings. To attain this goal the arrangement must must be constrained as in Claim 4, and therefore approximately $2n$ strings are required (Claim 5).

4.2 String layout

In Figure 7 we show a string layout which comes within two strings of the limits mentioned above. All the disks on the main diagonal (the pivot disks) belong to one of two strings. These two strings could be merged but we have chosen not to merge them so that the number of disks per string remains close to uniform.

The disks not on the main diagonal are laid out on strings which are diagonals perpendicular to the main diagonal. Each diagonal is drawn in a *wrapped* fashion. Starting with a disk in the bottom row, a line is drawn traveling in the northeast (up and to the right) direction. When it hits the right side of the array it wraps around to the left side. The disk immediately after the disk in the rightmost column is the disk in the leftmost column one row higher. From this disk the line continues along to the northeast until it terminates in the top row.

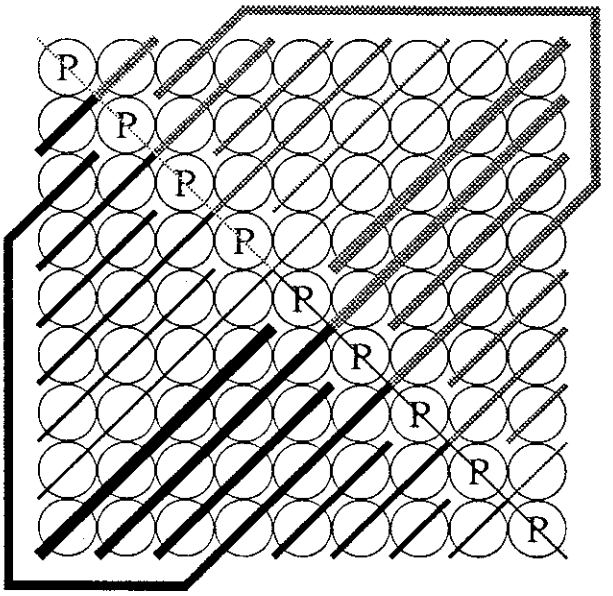

Figure 7: *An optimal string layout having all strings of the same length*

From a line we define two strings. All those disks on the line that are below the main diagonal are on one string and those that are above the main diagonal are on the other string. One of the disks on the line will be on the main diagonal. This disk belongs to one of the two strings running along the main diagonal.

If the pivot disks are placed on the main diagonal this layout is nearly optimal. It survives all pairs of string failures and it uses $2n + 2$ strings which is only 2 more than the $2n$ absolute minimum. It has the advantages that there are separate strings containing just pivot disks, and all of the strings but one are of the same length. One could choose to pack the n *virtual* parity disks on $n - 1$ actual disks, and have all strings the same length.

If the uniformity of the string length is not important, distributing the pivots among the other strings yields a layout which achieves the absolute minimum of $2n$ strings for n odd. This new layout (Figure 8) is the same as the one just described except that the two strings on the main diagonal are not used. The lines of disks described previously are divided into two strings in much the same way as they were before. One string is all the disks on the line that are above the main diagonal and the other string is all the disks on the line that are below *or on* the main diagonal. This layout uses $2n$ strings — half are of length $(n - 1)/2$ and half are of length $(n + 1)/2$. If n is even, the same arrangement can be used, and strings below the diagonal will have two parity disks.

Either of these two layouts is theoretically optimal in many ways. Their use in RAID will make it more reliable.

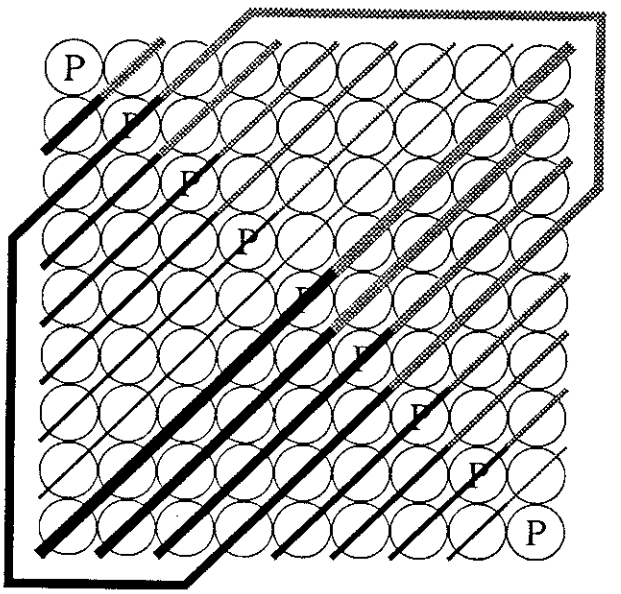

Figure 8: *An optimal string layout using the absolute minimum number of strings*

5 Calculating Reliability

The reliability of a particular RAID layout can be hard to evaluate. In the past researchers have relied on computer simulations to calculate reliability. However, the classification of bad squares characterizes those sets of disk failures that cause data loss. This, along with a simple model of disk repair times, allows us to calculate the reliability of a RAID layout to high precision without the use of a computer simulation.

To calculate the chance that, for instance, k disks have failed, we introduce a repair strategy which we call *scheduled repairs*. Here we assume that the entire disk array is repaired at regularly scheduled intervals. This is convenient for calculations, since now if disk failures are exponentially distributed, the probability of failure of a given disk on one interval is independent of any other interval. Furthermore, the probability of failure of the disk array in a given interval is exactly the probability that there is a bad arrangement of failed disks just before the repairs occur.

We can calculate the chance of data loss during a repair interval by calculating the chance that the set of disks which go bad during that interval include a bad square. An example of this calculation, estimating the mean time to data loss considering only disk failures, is outlined in Claim 6: We give an exact formula for the number of combinations of 3 failed disks and the pivot disk that include a bad square. We do the same for 4, 5, and 6 failed disks. We multiply the values generated by these formulas by the chance that a particular 3, 4, 5, or 6 disks fail and add the terms together. This gives a very good approximation of the probability that there is data loss. Table 1 gives some results.

n	*data disks*	*repair interval*	MTTDL *(years)*
5	16	daily	5,220,000
		weekly	106,000
9	64	daily	1,300,000
		weekly	25,600

Table 1: *Mean time to data loss of 2d-parity arrangement with one pivot. Mean time to failure of a single disk is 75K hours.*

The incorporation of information about string failures in the computation is complicated by the interactions between string failures and disk failures. Although the combinatorics is more complicated, the results are still reasonably precise. An outline of the calculations is given in Claim 7 and the results are given in Table 2.

	data disks	*repair interval*	MTTDL min *(years)*	MTTDL max *(years)*
5	16	hourly	297,000,000	313,000,000
		daily	517,000	547,000
		weekly	10,700	11,500
9	64	hourly	41,400,000	43,000,000
		daily	72,300	75,600
		weekly	1,410	1,690

Table 2: *Upper and lower bounds on mean time to data loss of 2d-parity arrangement with string failures. Mean time to failure of a single disk is 75K hours, and of a string is 250K hours.*

6 Conclusions

The 2d-parity arrangement of check disks removes bottlenecks but maintains high reliability of a disk array. The suggested string layout for the arrangement can withstand the failure of any two strings, the highest resiliance possible. Further, the layout uses the fewest strings possible, minimizing the expense for separate power supplies and disk controllers. If shorter strings are desired, they can be subdivided without weakening the layout significantly.

Mean time to data loss can be estimated explicitly by (1) classifying those collections of disks whose failure results in data loss, and (2) assuming a simple repair strategy. These

techniques give precise estimates by formula which were previously obtainable only by simulation.

These results leave open the issue of designing rectangular (rather than square) disk arrays. Furthermore, in application, it is desirable to have the strings be of a fixed size independent of the size of the disk array, for uniform design of controllers and power supplies. It is hoped the results in this paper could be generalized to these situations.

A Theorems and Claims

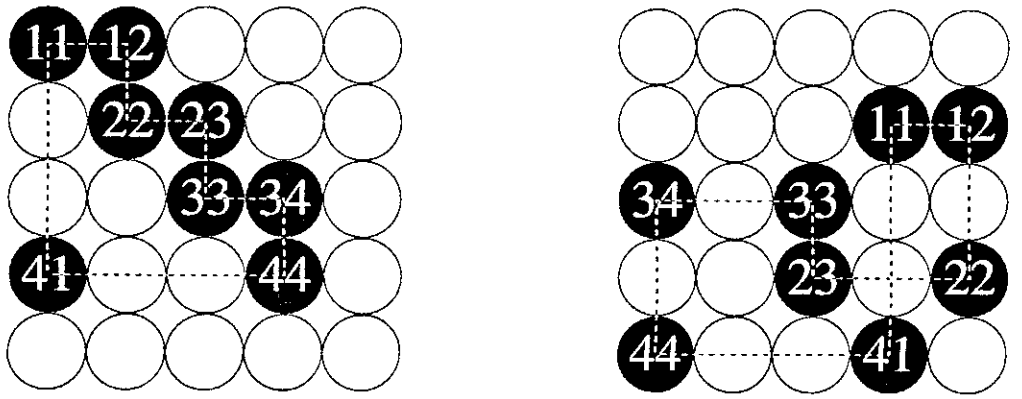

Figure 9: *Bad k-squares for $k = 4$. On the left is a bad square, and on the right is the same bad square after a permutation on the rows and columns.*

Theorem 1 *A set of failed disks, $S = \{D_{x_i,y_i}\}$, is a bad square for the complete 2d-parity arrangement if and only if it is some permutation of the rows and columns in*

$$S' = \bigcup_{0 \le l < k} \{D_{l,l}, D_{l,l+1 \bmod k}\}$$

for some k, $2 \le k \le n$ (see Figure 9).

Proof:

Failure of a bad square is fatal since each row and column of the bad square has exactly two failed disks: Toggling all of the bits in the failed disks would not affect any of the other disks in the array. Hence we cannot distinguish these two possible states of data to recover.

We will show by construction that every minimal set of failed disks, S, is a bad square. Choose a failed disk D_{x_1,y_1} from S. Since S is minimal, we must not be able to reconstruct this disk, and there exists another failed disk, D_{x_1,y_2} in the same row as D_{x_1,y_1}. Similarly

there exists a failed disk D_{x_2,y_2} in the same column as D_{x_1,y_2}. Continuing in this fashion, eventually we will repeat a column or a row; ie. for $j < i$ we will reach a disk D_{x_{i-1},y_i} with $y_i = y_j$, or a disk D_{x_i,y_i} with $x_i = x_j$. In the former case,

$$\{D_{x_j,y_j}, D_{x_j,y_{j+1}}, D_{x_{j+1},y_{j+1}}, \ldots, D_{x_{i-1},y_i}\}$$

form a bad square. In the latter,

$$\{D_{x_j,y_{j+1}}, D_{x_{j+1},y_{j+1}}, D_{x_{j+1},y_{j+2}}, \ldots, D_{x_i,y_i}\}$$

do. In fact, since S is minimal, the first case occurs, $j = 1$, and the x_i's and y_i's form the inverse of the permutation in the theorem. ∎

Corollary 1 *The number of bad k-squares is* $(n)_k^2/2k$.

Proof: $(n)_k^2$ is the number of ways to pick the ordered k rows and k columns. But this counts each permutation exactly $2k$ times, one for each possible start disk. ∎

Claim 1 *If every disk is a pivot on some plane, then there exists a string and disk combination whose failure is fatal, no matter what the layout of strings.*

Proof: Take a string, S, which contains any two disks D_1 and D_2. There exists a bad 2-square containing D_1 and D_2. A third disk in this 2-square will be the pivot for some plane, and failure of the fourth disk and the string S leads to data loss on this plane. ∎

Claim 2 *In complete 2d-parity, there exist a string and two disks whose failure is fatal, no matter what the layout of strings.*

Proof: Take a string, S, which contains two disks, and a bad 2-square containing these two disks. Failure of S and the other two disks in the bad square is fatal. ∎

Claim 3 *Complete 2d-parity requires* $n + 1$ *strings to protect against all possible failures of two strings.*

Proof: If any two strings contain $2n$ disks, then the failure of these disks is fatal: There are fewer disks remaining than there were data disks to start. Hence, the average length of a string must be strictly less than n, and there must be more than n strings. ∎

Claim 4 *If a string in a $2d-$parity arrangement with pivot disks on the main diagonal contains a disk D_{ij}, then it cannot contain another disk in rows i or j, nor in columns i or j.*

Proof: Suppose a failing string contains two disks in the same row, D_{ij} and D_{ik}. Either $i \neq j$ or $i \neq k$, so without loss of generality assume $i \neq j$. Then on the plane where D_{jj} is a parity disk, failure of the string containing D_{jk} leads to data loss.

Suppose, instead, a string containing disks D_{ij} and D_{ki} fails, and the disks are not in the same row, ie. $k \neq i$. Then on the plane where D_{ii} is a pivot, failure of the string containing D_{kj} leads to data loss. ∎

Claim 5 *2d-parity with pivot disks on the main diagonal requires at least $2n-1$ strings to protect against all possible failures of two strings. In particular, if n is odd, $2n$ strings are required; if, in addition, all the disks on the diagonal are to be on one string, then $2n+1$ strings are required.*

Proof: If we think of the weight of a disk, D_{ij} as two if $i \neq j$ and one if $i = j$, then Claim 4 implies a string cannot have weight exceeding n: Each disk D_{ij} in the string excludes other disks in the string from being in rows and columns i and j. A case analysis using this fact implies the claim. ∎

Theorem 2 *Failure of two strings in the layouts in Figures 7 and 8 in not fatal.*

Lemma 2.1 *If a complete 2d-parity disk array can be separated into two regions (A and B) by a monotonically non-decreasing or non-increasing line, such that no two disks have failed in the same row in region A and no two disks have failed in the same column in region B, then the outage is not fatal.*

Proof: Consider the trace of a bad square (given by Theorem 1) beginning at disk $D_{x_1y_1}$ in region A. Disk $D_{x_1y_2}$ in the same row must lie in region B, since no two disks have failed in the same row in region A. Similarly, disk $D_{x_2y_2}$ must lie in region A, etc.: The trace must alternately cross the dividing line of regions A and B by row and then by column (Figure 10) Since the dividing line is monotonic, the successive crossing points monotonically increase (or decrease), and the bad square cannot close. ∎

Lemma 2.2 *Failure of two strings in the layouts in Figures 7 and 8 is not fatal in a complete 2d-parity arrangement.*

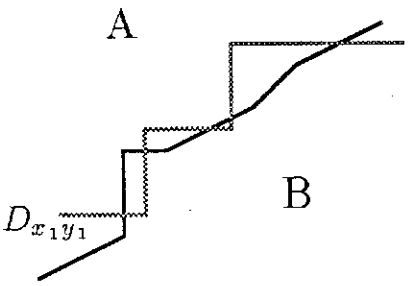

Figure 10: *A trace of a bad square (gray) with a dividing line (solid) as given by the lemma cannot exist.*

Proof: If the two strings (call them A and B) are not both on the same side of the main diagonal, then the previous lemma can be used directly since the main diagonal separates A from B. So assume the two strings are on the same side of the main diagonal. Each string is in two pieces A_1, A_2 and B_1, B_2, as in Figure 11 (Either A_1 or B_2 may be empty.) We consider two cases, and show a bad square cannot exist.

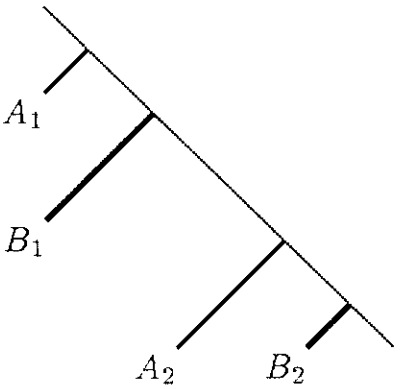

Figure 11: *Two strings below the main diagonal.*

Suppose a horizontal (or, by symmetry, vertical) line separates A_1 and B_1 from A_2 and B_2. Notice that any bad square must alternate between A and B. Without loss of generality assume that we are traversing the bad square in the direction in which column moves take us to A and row moves take us to B.

By Lemma 2.1 the bad square cannot lie entirely above or below the separating line. Hence, the bad square must cross the separating line between B_1 and A_2 since A_1 and B_2 have no column in common. This transition must be possible in both directions if the bad

square is to close. However, since the transition is only possible by column we know that it can never take go from A_2 to B_1.

If no horizontal or vertical line separates A_1 and B_1 from A_2 and B_2, then the construction guarantees that A_1 has no rows in common with B_1. Therefore A_1 (and similarly B_2) can be reconstructed, and Lemma 2.1 can be applied on the remaining down disks. ∎

Proof: (of Theorem 2) Call the two strings A and B. If one of the bad strings contains only parity disks, then the proof of the last lemma still goes through even if we assume all parity disks are out. Otherwise, a parity argument and the last lemma will show that any bad square cannot be made from the down disks plus one parity disk: A bad square leaving the parity disk and proceeding by row to A must then alternate between A and B, and return by column (and therefore from A) to the parity disk. But the construction guarantees that a parity disk does not have both a row and a column in common with a single string. ∎

Claim 6 *Let $s(k) = (n)_k^2/2k$ be the number of bad k-squares in an $n \times n$ disk array. (This is the equation given by Corollary 1.) For a 2d-parity layout with one pivot with independent disk failures of probability p in a repair cycle, the mean time to data loss is given by*

$$MTTDL = \frac{repair\text{-}interval}{\sum_{d=0}^{n} b(d) p^d (1-p)^{n^2-d}}$$

where $b(d)$ is the number of bad arrangements of d disk failures and is given by

$$\begin{aligned}
b(0) &= b(1) = b(2) = 0 \\
b(3) &= \frac{4}{n^2}\left[s(2)\right] \\
b(4) &= \frac{5}{n^2}\left[(n^2-4)s(2)\right] \\
b(5) &= \frac{6}{n^2}\left[s(3) + \binom{n^2-4}{2} s(2) - 4\binom{n}{2}\binom{n}{3}\right] \\
b(6) &= \frac{7}{n^2}\left[s(2)\binom{n^2-4}{3} + s(3)(n^2-6) - 18s(3)\right] \\
b(d) &\leq \binom{n^2}{d}
\end{aligned}$$

Proof: (omitted) ∎

Claim 7 *For the 2d-parity layout in Figure 5 with indepenedent disk failures of probability p and string failures of probability q in a repair cycle, the mean time to data loss is given by*

$$MTTDL = \frac{repair\text{-}interval}{\sum_{s=0}^{2n+2}\sum_{d=0}^{n^2} B(s,d)p^d(1-p)^{n^2-d}q^s(1-q)^{2n+2-s}}$$

where $B(s,d)$ is the number of bad arrangements of d disk failures and s string failures given by

$$\begin{aligned}
B(0,0) &= B(0,1) = B(0,2) = B(1,0) \\
&= B(1,1) = B(2,0) = 0 \\
B(0,3) &= n(n-1)^2 \\
B(0,4) &= 4n^5 - 7n^4 - 14n^3 + 33n^2 - 16n \\
B(1,2) &= \frac{13n^3 - 29n^2 + 17n - 1}{4}, n \text{ odd} \\
&= \frac{13n^3 - 29n^2 + 18n}{4}, n \text{ even} \\
B(s,d) &\leq \binom{2n+2}{s}\binom{n^2}{d}
\end{aligned}$$

Proof: (omitted) ∎

B Glossary

2d-parity Fundamentally, a 2d-parity arrangement is a disk with $m \times m$ data disks, plus one check disk for each row and each column. We consider the array to be $n \times n$ (where $n = m + 1$) , where the extra disk is a place marker for the pivot. Where data planes are involved, this pivot disk can be anywhere on the main diagonal. (See Figure 5.)

Complete 2d-parity A 2d-parity arrangement with one extra check disk containing the parity of all the data disks.

Bad Cycle The trace of a bad square given by Theorem 1. (See Figure 6).

Bad Square A minimal set of out disks causing data loss in a complete 2d-parity arrangement. A bad k-square is a bad square with failed disks in k columns and k rows. Notice minimality implies no data disks in a bad square can be recovered and that every set of fatal outages contains a bad square.

Data Plane Each disk is partitioned the same way into blocks. Corresponding blocks on all disks form a data plane. Each data plane is distinguished by its pivot disk.

$\mathbf{D_{xy}}$ The disk located in row x, column y in a disk array.

Fatal A set of disk and string failures is fatal if all data cannot be recovered from the remaining disks.

$\mathbf{m}$ The number of data disks in each row and column.

$\mathbf{n}$ $= m + 1$. This is the number of disks in a row including the check disk.

$(\mathbf{n})_{\mathbf{k}}$ $= n(n-1)(n-2)\cdots(n-k+1) = n!/(n-k)!$.

Pivot Disk The disk that defines where the check disks are. All disks in the same row (resp. column) as the pivot disk are column (resp. row) check disks. (See Figure 4)

Scheduled Repairs A repair strategy where the entire disk array is repaired at scheduled intervals, such as once a day at 4p.m.

String A group of disks whose failure is highly correlated, either because they have the same power supply, or the same controller.

Acknowledgements

We'd like to thank those who helped us with our research. Ethan Miller and David Patterson answered our hardware questions about RAID. They were able to guide us in our determination of what characteristics made a string layout good. We owe extra special thanks to Garth Gibson who was our sounding board. He listened to our ideas and gave good advice.

References

[PGK] Patterson, Gibson, Katz, *A Case for Redundant Arrays of Inexpensive Disks (RAID).*

[GHKKP] Gibson, Hellerstein, Karp, Katz, Patterson, *Coding Techniques for Handling Failures in Large Disk Arrays*, U.C. Berkeley Report No UCB/CSD 88/477, December, 1988.