SAND2013-0689C

RESAR Storage: a System for Two-Failure Tolerant, Self-Adjusting Million Disk Storage Clusters

Ignacio Corderi
University of California, Santa Cruz

Thomas M. Kroeger
Sandia National Laboratories

Thomas Schwarz
Universidad Católica del Uruguay

Darrell D. E. Long
University of California, Santa Cruz

Abstract

The organization of highly failure tolerant, but efficient storage clusters has not kept up with the increase in their scale, which is about to reach exabyte storage capacity. We present here a technique that abstracts the data layout of a two-failure tolerant storage system as a graph coloring model. Through emulation, we show that such a system scales to one million disks while providing an annual durability of 99.999999% (eight nines) with a storage overhead of only 20%. This durability is possible by distributing the reconstruction of a failed disk over many disks. In one emulation, for example, the work of rebuilding a one terabyte hard drive was evenly distributed across 459 disks and completed in less than four minutes with no disruption in service. This technique gives system designers and administrators fine-grained control to balance numerous system trade-offs. While we focus here on data resilience, our technique can be applied to optimize other system attributes such as load balancing and power consumption.

1 Introduction

Large scale storage clusters and cloud based storage are gaining widespread use as a result of the efficiencies and simplicity that they provide the end user. Unfortunately, most commercial clusters often rely on replication. The overhead of replication (100% or more) requires many extra disks relative to more efficient data encoding techniques such as RAID's parity striping. These extra disks cost in terms of maintenance and power. On the other hand RAID arrays provide some efficiencies with data encoding, but they suffer from significant recovery times to rebuild a failed disk and do not scale well as the number or size of the disks increases.

To address these issues we present, RESAR (Robust, Efficient, Scalable, Autonomous Reliability) a technique for managing large scale distributed resources. We use RESAR to manage parity encoding techniques across clusters from 250,000 to one million disks and are able to provide a resilence that is better than duplication and RAID6 with an overhead of only 20%. RESAR provides fast recovery from disk loss by distributing the work for recovery across hundreds of disks. In our emulations RESAR was able to recover from the loss of a one terabyte disk in under four minutes. In addition RESAR provides the ability for fine grained control of parity overhead while maintaining reliability guarantees that exceed most commercial systems that use replication.

In RESAR we divide each disk into a fundamental unit of data that we call a disklet (*e.g.* one terabyte could be divided into 51 20 GB disklets). Using these disklets, RESAR abstracts the data layout to a graph model where vertices represent parity and edges represent data. We then reduce the mapping of these disklets to specific hard disks to a graph coloring problem where we have as many colors as we have disks. The coloring algorithm is designed to distribute the work of rebuilding a failed hard disk across as many nodes as possible.

For example, in our emulations to recreate each failed disklet, nine systems (eight reliability group members, and one destination disk) would work in parallel streaming the 20 GB to calculate and store the data for each of the 51 lost disklets, for a total of 459 machines that are involved in the recovery of one disk dailure. The result is that while a RAID array may take hours to recover from a single disk failure in our emulations our RESAR cluster recovered from disk failure in under four minutes. As long as the cluster is large enough to distribute the recovery (typically more than a thousand disks) the expected recovery time is directly proportional to the selected disklet size and independant of the number or size of the disks in the cluster. In addition to fast recovery RESAR also provides the ability to do layout that is aware of multiple levels of resilience (*e.g.* disk, server, rack, site) and the ability to exercise fine grained control over how many data disks participate in a parity disklet.

The rest of the paper is organized as follows. Section 2 details RESAR. In sections 3 we present our emulation environment. Section 4 follows with the results from these emulations. Section 5 and 6 present a theoretical analysis. We discuss related and future work in sections 7 and 8 and conclude in section 9.

2 Design Distributed Data Resilience with Graphs

Here we present the use of graph models for data layout across a two-failure tolerant data encoding scheme. This technique we call RESAR, is used to create a one million disk storage cluster. We use *disklets* as the basic storage allocation element. A disklet represents contiguous storage on a disk, typically in the order of twenty to two hundred gigabytes. We show how each disklet is grouped into two reliability groups each with a parity disklet. We abstract this encoding as a graph where parity disklets are vertices and data disklets are edges of a graph, creating a multi-million node almost regular graph. We represent the assignment of a disklet to a disk as coloring a graph vertex or edge with a color that represents the disk. We find evaluating data layouts in terms of graph coloring easier and scalable. Our constraints enable a system to survive any two-element failure and distribute the work of recovery as broadly as possible, resulting recovery times of a few minutes instead of hours.

2.1 Disklets and Graph Representation

Data layout with RESAR involves two steps: First we place each disklet with data (a *data disklet*) in two reliability groups, each with n data disklets and one attached *parity disklet*. Parity is encoded as the *exclusive or* (XOR) of all the data disklets in the reliability group. Second, we assign disklets to disks in a manner that any two disklets in a reliability group are never collocated in the same disk. Note, we can also color with awareness to other properties such as distinct servers or racks but for this work we focus on disks.

In the language of Mathematical Design Theory, the reliability groups are blocks, subsets of the set of all disklets. We require that:
(1) Each disklet is in exactly two blocks.
(2) Each block contains exactly n elements.
(3) Two different disklets are in at most one block.
The last property is necessary to guarantee two-failure tolerance. If two disklets were in two different reliability groups, they would generate the same contribution to the two parity disklets. Therefore, if both were to fail, there is not enough data left to reconstruct the data in these disklets. A set of blocks with these properties is called a

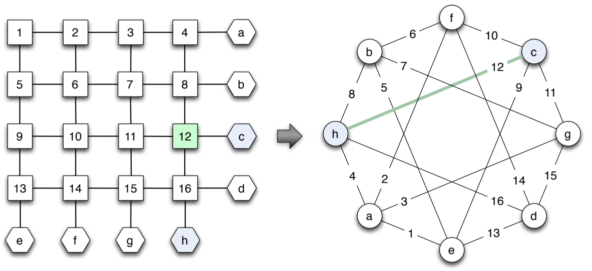

Figure 1: Left: small two-failure resilient array Right: its design-theoretical dual

configuration. The dual of a configuration is an n-regular graph, which we use to argue about layout. In this dual graph, the vertices are reliability groups and the edges are the data disklets that belong to the group. As each reliability group has exactly one parity disk, we achieve our graph representation, where parity disklets are represented by vertices and data disklets by edges.

Figure 1 gives an example for a small two-failure resilient array and its design-theoretical dual. The grid on the left shows data disklets numbered 1 to 16 and parity disklets given by a through h. The reliability groups are formed by the rows and columns. In the dual, the 8 parity disklets are the vertices and the data disklets are the edges. An edge connects two vertices if the corresponding data disklets is in the two reliability groups.

For our experiments, we use reliability groups of eight data disklets (*i.e.* $n = 8$). Correspondingly, the dual graph is eight-regular. We use a grid on a 4-dimensional torus for the graph. Our choice of $n = 8$ gives us a parity overhead of 20%.

2.1.1 Graph Coloring

Placing concrete disklets on disks into reliability groups is the same task as assigning abstract graph elements (the edges and vertices) to a disk. We represent this as *coloring* the dual graph. Our colors represent disks. If we want to survive rack failure or disk enclosure failures, we represent the disks in the rack or enclosure as a *palette*, a set of colors. Unlike classical graph coloring problems, we are coloring vertices and edges, we have many colors (as many as there are disks) and we want to use each color the same number of times. Like classical graph coloring problems, we cannot color adjacent elements with the same color. In fact, we strengthen this restriction by requiring that graph elements (vertices or edges, *i.e.* disklets) are not colored with the same color (or a color of the same palette) if they are very close. If we color a vertex with a color, we cannot reuse it on the edges emanating from it, and if we color an edge, we cannot reuse the color on its vertices nor on edges that share a vertex with it. As a consequence, if we have to reconstruct data on a failed disk, no disk has to read more than a single disklet, so that reconstruction workload is distributed.

While the coloring problem can be difficult for small graphs, a major result of our emulations is the ease in which a simple greedy algorithm that on occasion backtracks can solve it when we are having thousands of disks. While not linear, our algorithm comes very close.

2.1.2 Recoloring

Large storage systems are far from being static. Disk failure becomes a more than daily occurrence and changes in storage needs will change the number of disks and hence disklets. Our colored graph representing the data layout will change constantly. In response to these changes, we have to recolor parts of the graph. For instance, if a disk fails, then the data on the disklets in this disk need to be moved to different disks. We assume a certain number of unassigned, spare disklets in the system. When we recover the data on the disklets in the failed disk, we place them in other disks. In our dual representation, this corresponds to recoloring these disklets. The recoloring needs to satisfy the same restrictions as the initial coloring.

While this article focuses on recovery from disk failures, the concept of recoloring facilitates the administration of data movements for other reasons such as energy management (*e.g.* identifing stale data and collecting them on disks that we then can turn off) or performance (*e.g.* moving data from heavily used disks to save bandwidth.) In all cases, the graph representation allows the generation of simple algorithms that maintain two-failure tolerance and support other system goals.

2.2 Recovery

To recover a failed drive we must recover each of the disklets in the drive. Data disklets are in two different reliability groups. If all the disklets in either are available (or can be recovered), we can recover the data in the lost disklet. If both of them are available, we can decide which we want to use. We do not currently exploit this for optimization. Data in a lost parity disklet is regenerated by all data disklets in the reliability group.

In the graph representation, a disklet on a failed disk is marked as failed within the graph until the data is recovered when the disklet is given a new color corresponding to the location of the rebuilt disklet. Figure 2(a) shows a failed parity disklet and four data neighbours that are used to rebuild the parity disklet. On the right it shows how a failed data disklet has two possible choices for rebuilding, one for each reliability group. As long as the corresponding disks are available (or can be recovered), there is no data loss.

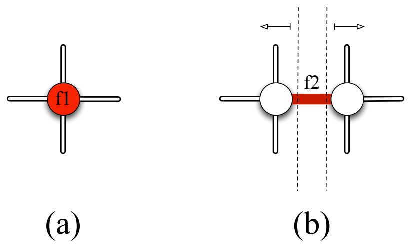

Figure 2: Left: failed parity disklet with its neighbours Right: failed data disklet with its neighbour

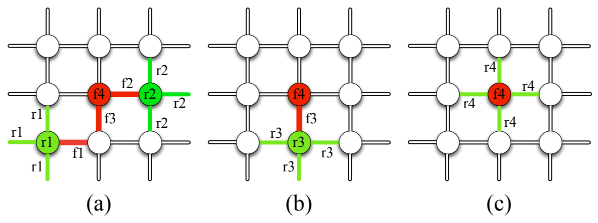

Figure 3: Cascading recovery of multiple failures in RESAR.

2.2.1 Recovery Patterns

Because of the highly interconnected nature of a RESAR cluster it turns out that in the vast majority of cases it can recover from the failure of more than two disks. A complete discussion is left to Section 5 but we show here how RESAR enables a cascade of recovery from failures. Figure 3 shows a case with four failures (marked f_1–f_4) that requires a cascade of recovery. In subfigure (a) the first two of these failures (f_1 and f_2) are data disklets that are reconstructed with the disklets labeled r_1 and r_2 resulting in subfigure (b). Then to recover f_3 disklets labeled r_3 are used. Finally in subfigure (c) we can calculate the parity disk labeled f_4 resulting in a complete recovery from four failures.

Arguments about failure tolerance are much easier in the graph than in the original primary design, as was previously observed [6]. Disk and sector failure induce a *failure pattern* in the graph. In spite of this well connected network there are still some patterns that are irreducible. These irreducible failure patterns describe instances of data loss. For an edge to be a part of an irreducible failure pattern, either the end-vertices also failed or at least one of the adjoining edges has also failed and is irreducible, or both. Therefore, minimal irreducible failure patterns are either a chain, Figure 4(a,b), or a cycle, Figure 4(c,d). The chain is a walk starting and ending at a failed vertex connected with failed edges in between. The cycle is and edge cycle in the sense of graph theory. The smallest minimal failure patterns are the *barbell* (Figure 4(b)) and the *triangle* (Figure 4(d)).

While these worst case scenarios may seem troublesome the truth is that RESAR's approach enables a strongly interconnected network of resilience across re-

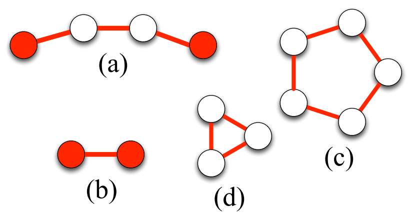

Figure 4: Irreducible failure patterns.

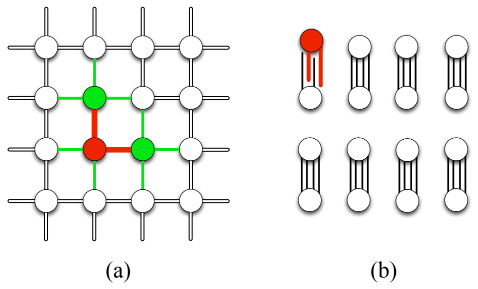

Figure 5: RESAR graphs (a) as compared to RAID6 (b).

liability groups. As we showed above a RESAR cluster can recover from a large number of failures in a cascade of disklet recovery. This is because RESAR enables a broader distribution of reliability groups across the complete cluster. Figure 5 shows the graph dual of a RESAR (a) and RAID6 (b) array with three node failures. In this case RESAR can recover but RAID6 would see data loss. When comparing this to the graphs of a RESAR cluster it is visually apparent why a RESAR cluster offers much greater resilience and more efficient use of resources in a distributed system.

2.3 RESAR System Dynamics

The two key parameters of a RESAR system are the total number of disks and the disklet size in the cluster. We now discuss how these parameters affect system dynamics.

For this work we focus on systems with disklets from 20 GB to 200 GB running on 1 TB hard disks, and clusters ranging from 250,000 to one million disks. If we assume that disks can sustain a read rate of 128 MB/s, a recovery of a 100 GB disklet will take approximately 800 seconds. If we recover all disklets on a failed disk in parallel and can neglect the time needed to discover the failure, then we only need 800 seconds to recover completely from the disk failure. If we have 1 TB hard drives (and use a 4 dimensional grid as a graph), then we recover 10 disklets, reading from 80 different disks and writing to 10 additional disks. If we shrink the disklet size to 20 GB, then we have to recover 51 disklets, read from 408 different disks, and write to 51 disks, but use only 160 seconds to do so.

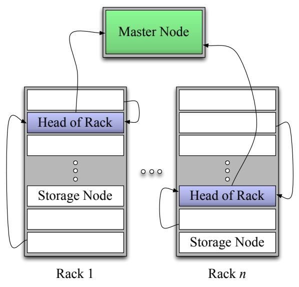

Figure 6: Physical layout with racks and servers

There are three reasons to use a lower bound for disklet size. First, if disklets are smaller, then there are many more of them and the administrative overhead of them increases. Second, fitting large resources like files into small containers (the disklets) becomes more difficult and the small spaces in a disklet that cannot be used because no resource fits into them will add up. Third, the number of different disks involved in a recovery is inversely proportional to the disklet size and has to be much lower than the total number of disks in the system for our coloring algorithm to work well.

In this work, we focus on systems with disklet sizes between 20 GB and 200 GB and scaling to a million 1 TB disks to reach an exabyte of total storage. For large clusters we intended to develop techniques for distributed coloring and administration.

3 Emulations

To better understand how RESAR would work at scale we used Sandia's Megatux platform [16] to emulate a RESAR cluster of over one million disks. Megatux was designed to use lightweight virtualization to study large-scale system behaviors. Its uses virtualization to emulate only the hardware and run the actual software stack. This approach gives us high-fidelity emulations and has proven very effective in large-scale system experimentation. Using high performance runs on the Jaguar supercomputer Megatux has been able to run as many as 4.5 million Linux VMs.

For this work we used a more modest cluster that emulated between 5,000 and 20,000 RESAR storage nodes. Each storage node emulates 50 one terabyte hard drives with read speeds of 128 MB/s.

3.1 Initial Layout

Currently we use a centralized algorithm for coloring the graph that runs on a single server called the Master Node. The Master node's responsibilities are storage node registration, layout creation and layout distribution. When a storage node starts it registers itself with the head of rack and the head of rack then forwards registrations to the master node. This allows us to scale registrations to hundreds of thousands of servers. Once the master has all the storage node registrations the *one time* layout process can be started.

Transmitting the whole layout does not scale very well. Instead to ensure each storage node can coordinate any needed recovery they are given the layout information of all of their disklets and the layout for all of the reliability group members of their disklets. In the graph dual this means a sever knows of all of it's disklets and every adjacent edge or vertex. This also avoids single point of failure and distributes the administration overhead for recovery. Distribution of layout is also aggregated by each head of rack. In our emulation each rack had 50 servers. The heads of racks receive the layout for all the storage nodes on the rack and then forwards the layout to each of them. On a storage server with 50 one terabyte drives with 20 GB disklets (2,550 disklets total) the layout is less than 1.5 MB of data. As a result the requirements on the storage nodes are independent of the cluster size and depend on how many disks the storage node has, this allows us to scale.

3.2 Recovery Process

The system is designed for scaling, parallelization and distribution. The recovery process is built around each disklet. When a disk fails a recovery process for every disklet is initiated. The load is then distributed to all surviving members in the reliability group.

When recovering a disklet, the storage node that owns it pseudorandomly chooses a recovery manager out of all the members of the disklets in the reliability group. The recovery manager requests up the chain to its head of rack what the new destination will be, builds a pipeline between the nodes and then informs each group member (and itself) which of the disklets must be used in the recovery and where to send it. Once a transfer request is completed the neighboring node informs the the recovery manager. When all neighbors finish and the disklet is written on its new location then the recovery manager informs up the chain to the head of rack and consequently to the master node that the recovery has finished.

By design the layout process ensures that when a disk fails every disk involved in the recovery has only one disklet to read. This means that for any failed disk we can recover as fast as we can read a disklet from the slowest hard drive involved. With 20 GB disklets on a 1 TB hard drive with reads of 128 MB/s, 459 disks will recover have an expected recovery of approximately 2 minutes and 40 seconds. 100 GB disklets will expect to recover in approximately 13 minutes and 20 seconds.

The network bandwidth requirements on the nodes depends on where in the pipeline the node is located. In Figure 7 the bandwidth requirement for nodes 2, 7 is three times the slowest read speed (2 inbound streams and 1 outbound), nodes 3, 9 and 10 need double the slowest read speed (1 inbound and 1 outbound). For all other nodes the network bandwidth is simply the slowest disk speed. Where each node sits on the pipeline is decided by the recovery manager based on the physical layout and available resources. If two nodes in the recovery process are on the same rack then they will form an edge on the pipe (nodes 8 and 3 could be on the same rack). We envision a system that would load balance and allows nodes that are already very loaded to participate as leaf nodes (nodes 8, 6, 5, 4 only have to transfer the contents and does not need to XOR), requiring them only to forward their data.

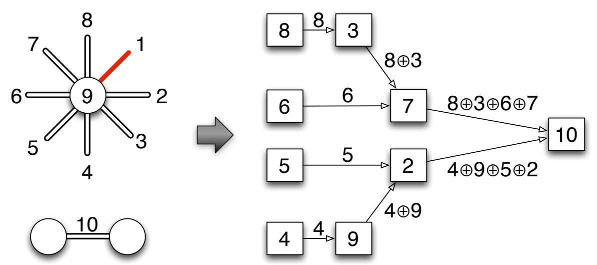

Figure 7: Top left: failed data disklet and one of its reliability groups. Bottom left: destination disklet. Right: recovery pipeline

To recover a 1 TB drive in 2 minutes and 40 seconds the total bandwidth on the system would be 51 GB/s (eight recovery edges at 128 MB/s $\times$ 51 disklets). With 1 million drives in 20,000 servers grouped in 400 racks, assuming an even work distribution each rack would have a bandwidth requirement of 130.6 MB/s. In short, modest data center interconnect could easily handle these requirements.

The size of the disklets should be dimensioned according to the expected size of the infrastructure but it is not a hard requirement. If the storage system has small number of drives compared to the total required to recovery from a failure the layout process would relax its restriction on using all different disks to recover from a disk failure. Essentially slowing down the recovery process to as fast as the entire system allows.

3.2.1 Drives

In our emulations we accounted for disk reads during recovery assuming a 128 MB/s IO rate. We looked at online reviews of hard drives performance and found several products that meet our assumption. A Seagate Barracuda 1 TB 3.5 inch Hard Drive can sustain rates of more than 128 MB/s for less than 100 USD. In order to

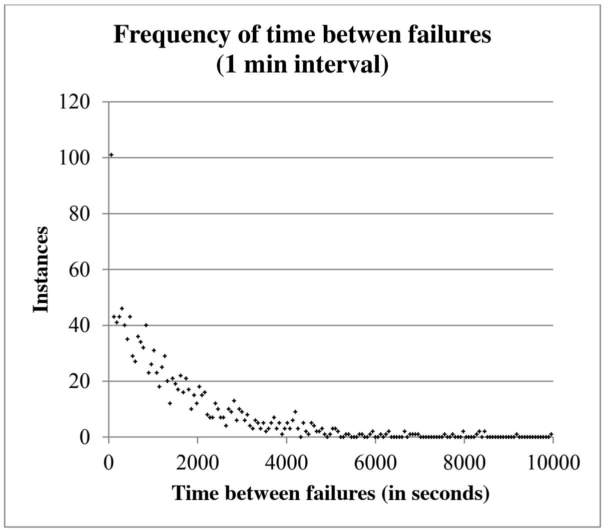

Figure 8: System wide time between failures on 500,000 disks.

speed up the emulation the actual reading time is simulated and depends on the current emulation clock multiplier. Each emulated hard drive handles its own read requests. If during the emulation multiple failures required to read the same drive we assumed a worst case where each request would have to wait until the previous one is served.

Every 200 ms to 2 seconds each storage server goes through all the disks he manages and decides based on pseudo randomly drawn numbers if a given disk should fail. The probability of a disk dying between the last time it was checked the the current check is a uniform probability throughout the year. The whole system behaves like a poisson process with exponential times between failures as shown in Figure 8. Pinheiro *et al.* [20] show failure rates between 2% and 8% changing with the hard drive age. Given that our simulation run for months at a time or at most one year, we simplified our failure model to a flat 4% annual disk failure rate.

3.3 Emulation Clock

We used a clock multiplier to enable our emulations to cover system behavior over longer periods of time. Typically, we use a multiplier of 30 (*i.e.* 1 second or wall clock accounts for 30 seconds of emulation time) but in one case, a clock multiplier of 600 (*i.e.* 1 second covers 10 minutes) to enable experiments that cover over a year of system behavior. As our clock multiplier grew our emulations incurred an additive error because of the workload on the busy emulation nodes. The result is that our emulated recovery times shows a recovery time greater than what one would expect in real life. Figure 9 shows emulated recovery times as the time multiplier varies, as the multiplier approaches a value of one we see the av-

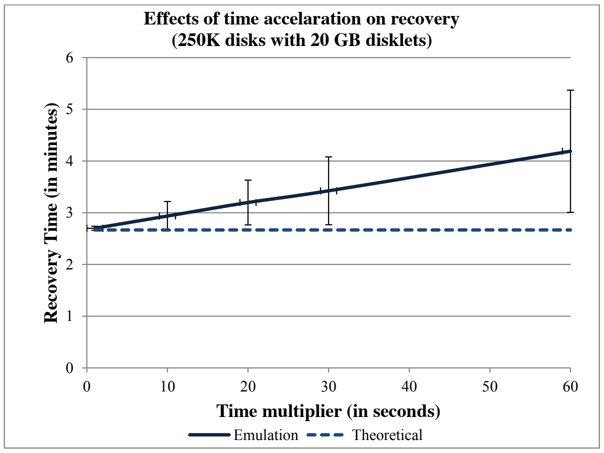

Figure 9: Average recovery time over 100 failures as the emulation time multiplier increases with 95% confidence intervals.

erage recovery approach to the expected optimal recovery time and we see less variance. The reason behind this emulation error is that at a high clock multiplier, each time a storage node takes a few extra hundred milliseconds to finish processing the recovery notification messages the emulation clock would see a whole extra minute. Even so these worst case recovery times are still orders of magnitude better than the recovery of comparable RAID systems.

4 Results

Here we present the results of our emulations of RESAR storage clusters. Our emulations spanned many key system parameters and showed that RESAR presents a stable system that can efficiently distribute data recovery across many disks enabling recovery from a one terabyte disk failure in under four minutes. We show how recovery time is linear with disklet size for reasonable disklet sizes. Our bemulations of RESAR clusters that show over a year of stable behavior and scales of up to one million disks.

4.1 Initial Data Points

Initially to demonstrate the system behavior we present our emulations of a RESAR cluster running with 500,000 disks and a disklet size of 20 GB. Figure 10 shows the recovery time for disk failures across this emulation. We used a clock multiplier of 30 and ran this emulation for 20 days of emulation time. This graph shows a recovery that is typically well under four minutes. These results show that our data layout algorithm was successful in distributing the workload of disk recovery.

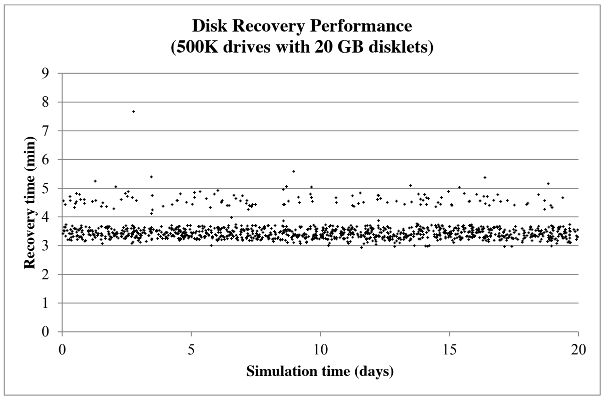

Figure 10: Disk recovery time on a system with 500,000 disks and 20 GB disklets, with a clock multiplier of 30.

4.2 Disklet Size and Recovery

Next we explored how variations in disklet size impact recovery times. To examine this we emulated a system of 500,000 disks with disklet sizes of 20 GB, 50 GB and 100 GB. Figure 11 shows the recovery times as disklet size varies. Each data point represents an emulation of a RESAR cluster for running for one week of emulation time at a clock multiplier of 30. This figure includes error bars that represent 95% confidence intervals (barely discernible due to negligible variance). These results closely track our theoretical models and show how, for reasonable parameters, disk recovery time is directly proportional to disklet size. Additional data for emulations of a cluster with 200 GB disklet was consistent with this these results but was left out to make the graph more readable.

As disklet size is reduced we are able to distribute recovery across more disks causing a faster recovery. In practice as disklet sizes become smaller factors such as network bandwidth and number of disk in the system would come into play. As mentioned before, for this work we focus on reasonable disklet sizes from 20 GB–200 GB where disk I/O is the dominant factor.

4.3 Scale and Long Term Behavior

To look at RESAR across large scales and longer term behavior we ran emulations that covered over a year of emulated time and clusters of 250,000, 500,000 and one million disks. For these tests we used a clock multiplier of 600 (*e.g.* one second covers 10 minutes) and a disklet size of 100 GB. Figures 12 through 14 show the recovery behavior of these system as we double the storage capacity. In all emulations, the disk recovery time clustered in a range between 17 minutes and 34 minutes for 100 GB disklets regardless of the number of disks on the system. The expected recovery time for 100 GB disklets is 13 minutes 20 seconds. The recovery time in Figures 12

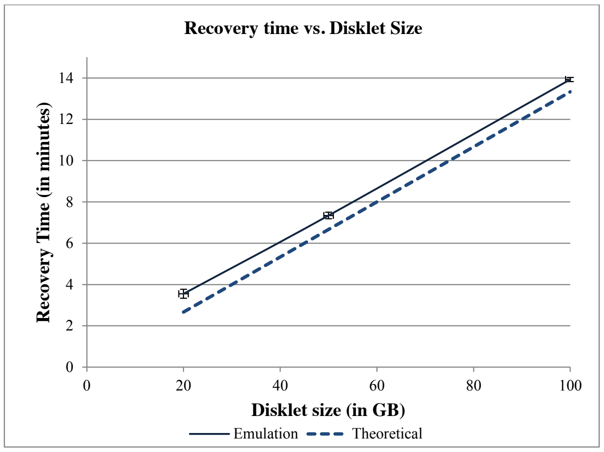

Figure 11: Disk recovery times on a cluster of 500,000 disks with a clock multiplier of 30. Error bars (barely discernible due to negligible variance) show 95% confidence intervals and each data point represents one week of emulation time.

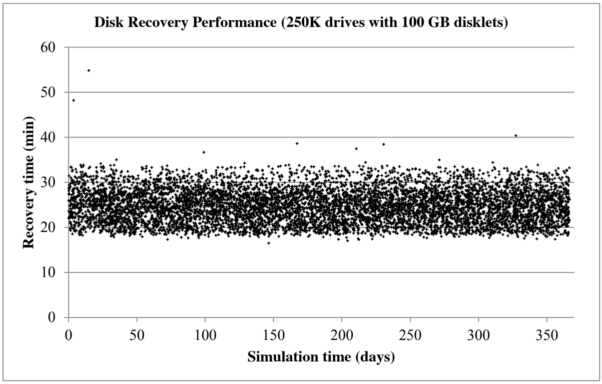

Figure 12: Disk recovery time on a system with 250,000 disks and 100 GB disklets for one year, with a clock multiplier of 600.

to 14 consistently bottoms out at the expected theoretical recovery rate. From these graphs and the clock multiplier discussion on Section 3.3 we can conclude that the wider range of recovery times is simply a result of additive error from the clock multiplier.

Finally, we note that a system with four times as many disks saw a four times as many disk failures as illustrated by the higher density of data points in figure 14. Our RESAR cluster show a consistent behavior across this increase in failures and shows stable behavior for one year in each emulation.

5 Failure Resilience

Comparison of failure tolerance among different organizations is difficult [8], and calculation of expected annual loss rate is beyond our mathematical capabilities at least for RESAR itself. To yield comparable results,

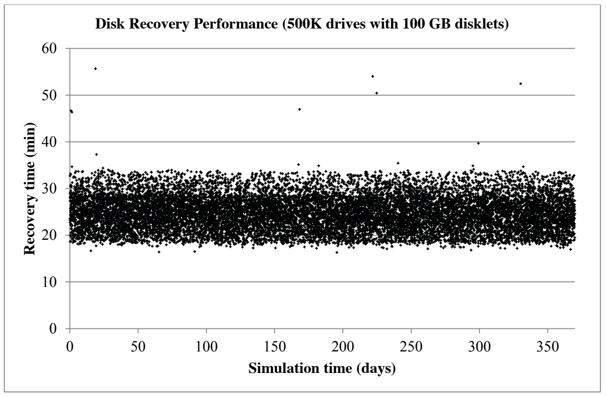

Figure 13: Disk recovery time on a system with 500,000 disks and 100 GB disklets for one year, with a clock multiplier of 600.

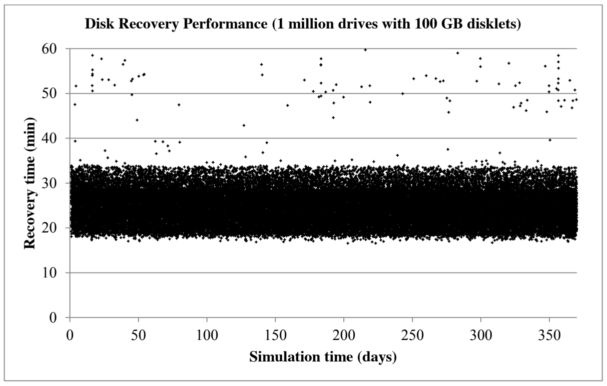

Figure 14: Disk recovery time on a system with 1,000,000 disks and 100 GB disklets for one year, with a clock multiplier of 600.

we restrict ourselves to calculating the *robustness*, *i.e.* the probability of not suffering dataloss, given three disk failures in the ensemble. To make numbers comparable, we assume the same usable storage capacity.

The simplest to administer, but also the most costly means of achieving failure tolerance is replication. If we replicate, then we have $2n$ disks and loose data if a pair of mirrored disks has failed. Among the $\binom{2n}{3}$ patterns of three lost disks, we get to pick a pair of mirrored disks (n possibilities) and an additional disk ($2n-2$ possibilities) giving us the probability of data loss of

$$p_{\text{DL Dup}} = \frac{n(2n-2)}{\binom{2n}{3}}$$

The formula for triplication is simpler, as we get to pick just one set of mirrored disks:

$$p_{\text{DL Tri}} = \frac{n}{\binom{3n}{3}}$$

If we use error-control codes, then the simplest organization is a *distributed RAID Level 6*, where we partition

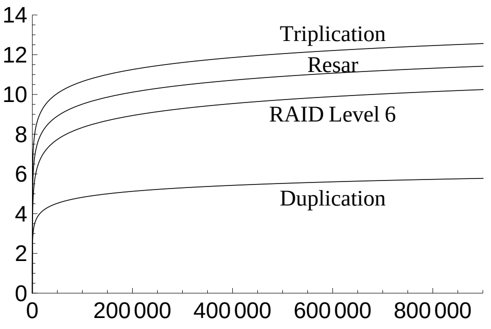

Figure 15: Robustness (number of nines) against simultaneous failure of three disks as a function of the usable storage capacity, given by the number of disks.

the disks into reliability groups each one with k data disks and two parity disks. This gives us a total of $n(k+2)/k$ disks. We suffer data loss if all three of the failed disks are located in the same reliability group. This gives us

$$p_{\text{DL DR6}} = \frac{n}{k}\binom{k+2}{3}\binom{n(k+2)/k}{3}^{-1}$$

We are using RESAR with a graph that is based on a grid in a four-dimensional torus. To maintain comparibility, we assume that the disklets take up the whole disk. The only three-failure pattern is the barbell, since in such a graph, there are no cycles consisting of three edges. The number of barbells is equal to the number of data disks. The total number of disks is $n(k+2)/k$, as it has the overhead of the distributed RAID Level 6.

$$p_{\text{DL RESAR}} = n\binom{n(k+2)/k}{3}^{-1}$$

For the same storage overhead, a RESAR layout is 15 times more robust than the distributed RAID Level 6 layout. We compare the robustness against three simultaneous failures in Figure 15, where we give the number of nines of the probability of all data surviving in the y-axis and the usable storage capacity measured in the number of disks in the x-axis. As we can see, RESAR is about half-way between triplication and distributed RAID Level 6. This comparison is however heavily biased against the more storage efficient layouts, since they have many fewer disks that can possibly fail.

6 Failure Distribution

To gain insight into the number of failed disks in a large disk array, we can use a Markov model that makes the simplifying assumptions of constant failure rates and repair times. We characterize the system by the states 0, 1, *etc...*, where the number indicates the number of disks currently failed. To simplify, we assume that failed disks are replaced ("repaired") at a constant rate ρ and that

Table 1: Predicted probability of having a certain number of disk failures in a cluster with 250,000, 500,000, 1,000,000 and 10,000,000 disks of 1TB capacity organized as a RESAR layout with 100 GB disklets.

Failures	250,000	500,000	1,000,000	10,000,000
0	77.6%	60.2%	36.3%	0.0%
1	19.7%	30.6%	36.8%	0.0%
2	2.5%	7.8%	18.7%	0.2%
3	0.2%	1.3%	6.3 %	0.7%
4	0%	0.1%	1.6 %	1.7%
5	0%	0%	0.3 %	3.5%
6	0%	0%	0%	6.0%
7	0%	0%	0%	8.6%
8	0%	0%	0%	10.9%
9	0%	0%	0%	12.4%
10	0%	0%	0%	12.5%
11	0%	0%	0%	11.5%
12	0%	0%	0%	9.7%
13	0%	0%	0%	7.6%
14	0%	0%	0%	5.5%
15	0%	0%	0%	3.7%

disks fail at a constant rate λ. The total number of disks is n. The state transitions are failure transitions from State s to State $s+1$ taken with rate $(n-s)\lambda$ and repair transitions from State s to State $s-1$ with rate $s\rho$. We assume that we do not have failure transitions, so that the model is an Erlangian birth-death system. We are interested in the probabilities p_i to be in State i when the system is in equilibrium. We have the Kolmogorov equations for the steady state

$$\begin{aligned} n\lambda p_0 &= \rho p_1 \\ (n-s+1)\lambda p_{s-1} + (s+1)\rho p_{s+1} &= ((n-s)\lambda + s\rho)p_s \end{aligned}$$

We can deduce (and prove by induction) that

$$p_s = \frac{n-s+1}{s}\frac{\lambda}{\rho}p_{s-1}$$

Since the ratio $\kappa = \lambda/\rho$ is quite small as disks lasts for long time and replacement is relatively quick, the p_s converge quickly to zero for $s \to \infty$, giving us excellent numeric solutions even though no simple closed form expression exists.

We give the predicted probabilities of the system having currently a certain number of disk failures in Tabel 1. As we can see, a large enough system will almost always have a certain number of disks under repair. The median for the expected number of failures is $\frac{\lambda N-\rho}{\lambda+\rho}$, as we can see from the recurrence relation. The fraction of disks used in recovery work approaches asymptotically $\frac{\kappa}{1+\kappa}$, which for reasonable disk repair and failure rates is close to κ.

7 Related Work

Overcoming the restricted bandwidth of a single disk was the original driving force behind disk arrays. For example, Cray research striped files over disks in the eighties [12]. Redundant Arrays of Inexpensive (later Independent) Disks (RAID) added a single, distributed parity disk to striping [4, 7, 18, 19]. To shorten recovery times, disk arrays of that time usually included one or more spare disks that allowed instant recovery once a failure was detected. Distributed sparing distributes not only the parity disk but also the spare disk [13]. Researchers quickly realized that increasing the number of disks gives opportunities for additional failure resilience. Declustering forms virtual RAID Level 4 stripes from blocks (corresponding to our disklets) on different disks [10, 11, 14, 15, 17, 21]. In the case of a disk failure, declustering distributes the reconstruction workload equally among all disks, giving better performance in the presence of a failed disk, leading to shorter recovery time and thus closing the window of vulnerability [24] where an additional failure results in data loss. The problem of laying out a disk array with a moderate number of disks (fewer than 100) translates into problems of Combinatorial Design theory, a branch of Combinatorics. A variety of layouts have been proposed, which achieve optimality in some sense [1, 2, 22, 23].

The possibility of additional disk failures during the window of vulnerabilities, and especially the impact of "bad blocks" discovered only during data recovery operations has fueled research into using erasure codes with two parities or placing a disk block into two different reliability stripes. The earliest proposal of the latter type comes from Hellerstein *e*t al. [9]. Erasure codes that use only exclusive-or operations have also been developped such as Even-Odd [3], X-codes [25], Row-Diagonal Parity (RDP)-codes [5], among others. This type of work continues.

8 Future Work

We presented RESAR and explored its failure tolerance. A large number of important issues are left unexplored. In the area of reliability, disk failures are not the only source of data loss. Latent sector faults are often only discovered when disks are accessed for recovery purposes and can lead to data loss. We did not model this yet. Second, disks are not the only components that can fail. It is certainly reasonable to protect against temporary or complete loss of all disks in a rack or an enclosure unit, because vibration, lack of cooling, etc. can damage all disks in it.

The fact that all data is protected by two different reliability groups allows us to use two disjunct sets of disks in order to recover a disklet. As disks experience accesses by clients, the two ways of recovering a disklet (and the many different ways of recovering the data in a lost disk) do not have the same costs.

The graph representation gives a more intuitive abstraction for programming tasks such as creating zero traffic disks that can be turned off for energy savings or that can be decomissioned, integrating new disks into the cluster, or redistributing disk load for better client access times and to manage network load. However, we have not programmed and emulated these actions.

Currently, our emulations use a single command module. As storage systems scale towards exabytes, control needs to be distributed and itself protected against failure. While this task seems to be simple engineering, we have not performed and tested it. Additionally, we intend to enhance our emulations to include IO workloads and examine write performance.

An architecture based on a distributed RAID Level 5 or 6 lends itself to more efficient forms of parity logging as the updates perculate only within the RAID group. In our layout, parity logging is more challenging. An efficient implementation might need to have the parity disklet become virtual so that an update to a data disklet yields a relocation of the two corresponding parity disklets, or for a busy parity disklet to live in RAM. These aspects has not yet been sufficiently considered in our work.

9 Conclusions

We have presented RESAR, a technique for dealing with large scale distributed resource management and have shown how this technique can be used to create a resilient and efficient storage cluster scaling from thousands of disks to millions of disks. Our model for expressing data layout as a graph coloring problem present great promise for large cluster efficiencies with fine grain control over parameters for reliability and overhead. Our analysis shows of how a RESAR cluster would compare to the reliability of techniques like replication and RAID. Mathematically RESAR is 15 times more resilient than RAID 6 with the same storage overhead and nearly as resilient as triplication with far less overhead. RESAR by it's intrinsic nature provides a more resilient structure for reliable large scale data. In addition this system offers great potential for fine grained control of numerous system parameters such as parity overhead and reliability guarantees.

Using Megatux emulation system we have run the actual software stack that would manage a RESAR cluster and have shown the system to be efficient in compute and memory requirements. Our emulations verified our analytical results for a stable and scalable system a range of cluster and disklet sizes. The distributed and parallel nature of the recovery mechanism allows the system to recover from a disk failure in as little as four minutes. Far ahead of recovery times for systems with comparable overhead.

Acknowledgments

This work was supported in part by the National Science Foundation under award CCF-1219163, and by Sandia National Laboratories under LDRD 158479. We are grateful for the support and help of John Floren, Ian Adams, Ahmed Amer and Aleatha Parker-Wood. 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.

References

[1] ALVAREZ, G. A., BURKHARD, W. A., AND CRISTIAN, F. Tolerating multiple failures in RAID architectures with optimal storage and uniform declustering. In *Proceedings of the 24th Int'l Symposium on Computer Architecture* (Denver, CO, June 1997), ACM, pp. 62–72.

[2] ALVAREZ, G. A., BURKHARD, W. A., STOCKMEYER, L. J., AND CRISTIAN, F. Declustered disk array architectures with optimal and near-optimal parallelism. In *Proceedings of the 25th Int'l Symposium on Computer Architecture* (1998).

[3] BLAUM, M., FARRELL, P. G., AND VAN TILBORG, H. C. A. Array codes. In *Handbook of Coding Theory*, V. S. Pless and W. C. Huffman, Eds., vol. 2. North-Holland, Elsevier Science, 1998.

[4] CHEN, P. M., LEE, E. K., GIBSON, G. A., KATZ, R. H., AND PATTERSON, D. A. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys 26*, 2 (June 1994), 145–185.

[5] CORBETT, P., ENGLISH, B., GOEL, A., GRCANAC, T., KLEIMAN, S., LEONG, J., AND SANKAR, S. Row-diagonal parity for double disk failure correction. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST '04)* (2004), pp. 1–14.

[6] CORDERI, I., SCHWARZ, T., AMER, A., LONG, D. D. E., AND PÂRIS, J.-F. Self-adjusting

two-failure tolerant disk arrays. In *Proceedings of the 5th International Workshop on Petascale Data Storage (PDSW10), held in conjunction with SC2010* (Nov. 2010).

[7] GIBSON, G. A. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*. PhD thesis, University of California at Berkeley, 1990.

[8] GREENAN, K., PLANK, J., AND WYLIE, J. Mean time to meaningless: MTTDL, markov models, and storage system reliability. In *Proceedings of the 2nd USENIX conference on Hot topics in storage and file systems* (2010), USENIX Association, pp. 5–5.

[9] HELLERSTEIN, L., GIBSON, G. A., KARP, R. M., KATZ, R. H., AND PATTERSON, D. A. Coding techniques for handling failures in large disk arrays. *Algorithmica 12* (1994), 182–208.

[10] HOLLAND, M., GIBSON, G. A., AND SIEWIOREK, D. P. Fast, on-line failure recovery in redundant disk arrays. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing (FTCS '93)* (1993), pp. 422–431.

[11] HOLLAND, M., GIBSON, G. A., AND SIEWIOREK, D. P. Architectures and algorithms for on-line failure recovery in redundant disk arrays. *Journal of Parallel and Distributed Databases 2*, 3 (July 1994), 795–825.

[12] JOHNSON, O. Three-dimensional wave equation computations on vector computers. *Proceedings of the IEEE 72*, 1 (Jan. 1984), 90 – 95.

[13] MENON, J., AND MATTSON, D. Distributed sparing in disk arrays. In *Proceedings of Compcon '92* (Feb. 1992), pp. 410–416.

[14] MERCHANT, A., AND YU, P. Design and modeling of clustered RAID. In *Digest of Papers of the 22nd International Symposium on Fault-Tolerant Computing* (1992), IEEE, pp. 140–149.

[15] MERCHANT, A., AND YU, P. Analytic modeling of clustered RAID with mapping based on nearly random permutation. *IEEE Transactions on Computers, 45*, 3 (1996), 367–373.

[16] MINNICH, R., AND RUDISH, D. Ten million and one penguins, or, lessons learned from booting millions of virtual machines on hpc systems. In *Proc. Workshop on System-level Virtualization for High Performance Computing in conjunction with EuroSys* (2010), vol. 10.

[17] MUNTZ, R. R., AND LUI, J. C. S. Performance analysis of disk arrays under failure. In *Proceedings of the 16th Conference on Very Large Databases (VLDB)* (Brisbane, Queensland, Australia, 1990), Morgan Kaufmann, pp. 162–173.

[18] PATTERSON, D., CHEN, P., GIBSON, G., AND KATZ, R. Introduction to redundant arrays of inexpensive disks (RAID). *Spring COMPCON89* (1989), 112–117.

[19] PATTERSON, D. A., GIBSON, G., AND KATZ, R. H. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data* (1988), ACM, pp. 109–116.

[20] PINHEIRO, E., WEBER, W.-D., AND BARROSO, L. A. Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)* (Feb. 2007).

[21] SCHWABE, E., AND SUTHERLAND, I. Improved parity-declustered layouts for disk arrays. In *Proceedings of the sixth annual ACM symposium on Parallel Algorithms and Architectures* (1994), ACM, pp. 76–84.

[22] SCHWARZ, T., AND BURKHARD, W. A. Almost complete address translation (ACATS) disk array declustering. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing* (New Orleans, Oct. 1996), IEEE, pp. 324–331.

[23] SCHWARZ, T., STEINBERG, J., AND BURKHARD, W. A. Permutation development data layout (PDDL). In *Proceedings of the 5th Int'l Symposium on High-Performance Computer Architecture (HPCA-5)* (Jan. 1999), IEEE, pp. 214–217.

[24] XIN, Q., MILLER, E. L., SCHWARZ, T. J., LONG, D. D. E., BRANDT, S. A., AND LITWIN, W. Reliability mechanisms for very large storage systems. In *Proceedings of the 20th IEEE / 11th NASA Goddard Conference on Mass Storage Systems and Technologies* (Apr. 2003), pp. 146–156.

[25] XU, L., AND BRUCK, J. X-Code: MDS array codes with optimal encoding. *IEEE Transactions on Information Theory 45*, 1 (1999), 272–276.