2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems

PERSES: Data Layout for Low Impact Failures

Avani Wildani*, Ethan L. Miller†, Ian F. Adams‡, and Darrell D.E. Long†
*The Salk Institute, †University of California: Santa Cruz, ‡EVault Inc.

***Abstract*—Growth in disk capacity continues to outpace advances in read speed and device reliability. This has led to storage systems spending increasing amounts of time in a degraded state while failed disks reconstruct. Users and applications that do not use the data on the failed or degraded drives are negligibly impacted by the failure, increasing the perceived performance of the system. We leverage this observation with PERSES, a statistical data allocation scheme to reduce the performance impact of reconstruction after disk failure.**

PERSES reduces degradation from the perspective of the user by clustering data on disks such that data with high probability of co-access is placed on the same device as often as possible. Trace-driven simulations show that, by laying out data with PERSES, we can reduce the perceived time lost due to failure over three years by up to 80% compared to arbitrary allocation.

I. INTRODUCTION

Over time, failure events are inevitable in large disk-based storage systems [1], [2]. Even in failure events where data is not lost, there is a system-wide cost for accessing data on the failed disk. The failed disk is inaccessible, and other disks and the interconnecting network links may be saturated. This cost is increasing as growth in disk size outpaces growth in disk, CPU, and network bandwidth [3], [4]. While a 100 GB drive that can be rebuilt at 50 MB/s is fully operational in about half an hour, a terabyte drive at 50 MB/s can take up to six hours to be back at full bandwidth. Additionally, during a rebuild there is often significant network overhead that can have negative side effects on disks that are not even involved in the rebuild.

We examine the issue of excessive rebuild time from the perspective of *projects*: groups of records that are frequently accessed together. If a failed disk contains even a small amount of critical data for multiple projects, all of those projects could suffer degraded performance until the rebuild is completed. Examples include developers losing a header file that a build depends on, a high performance system that loses an input and misses its time slot, or a cloud provider that breaks a performance SLA.

If, however, the failure occurs on a device that is not actively being accessed, it has almost no productivity cost: it is the proverbial tree fallen in a forest. PERSES is a data allocation framework designed to decrease the impact of device failures on the productivity and perceived availability of a storage system. We name our system after PERSES, the Greek titan of cleansing destruction, because it focuses destruction in a storage system to a small number of projects so that others may thrive. Our goal with PERSES is to isolate failures away from as many projects as possible.

Multi-user cloud storage systems, a primary target of PERSES, are caught in the data deluge. According to the 2013 IDC report, digital content is expected to grow to 4 zettabytes by the end of 2013, which would be a 48% increase over 2012 [5]. Cloud applications such as web hosting and document sharing maintain SLAs that guarantee low response time to users. In this environment, the cloud provider saves money by isolating failures to, and thus breaking SLAs with, as few customers as possible. While they experience more degradation, the marginal cost of additional slow response time is often lower for users once any slowdown has occurred.

Exacerbating this degradation, many systems sacrifice rebuild speed in exchange for increased reliability. These systems typically place data evenly across disks in order to distribute and thus dilute the impact of failure events. When a failure event does necessitate a rebuild, however, the rebuild can impact a wide array of people or projects even though each could have only lost a very small amount of data. By combining existing reliability measures with selectively laying out data according to group membership, we can isolate failures so that they impact fewer users or projects, increasing the perceived availability of the system.

Other use cases that are disproportionately affected by partial data degradation include compilation and select scientific workloads. Compiling large code bases typically involve multiple dependencies, and developers working with large code bases often have to wait to compile until all of the files they need are available. An unlucky failure can halt the compilation of large swathes of code for want of one file. Similarly, scientific data analysis can rely on a small number of key files such as format descriptors or configuration files that slow the analysis of terabytes of data. According to the Tech Lead of Facebook's HBase engineering team: "Even losing a single block of data incurs a high fixed cost, due to the overhead of locating and recovering the unavailable data. Therefore, given a fixed amount of unavailable data each year, it is much better to have fewer incidents of data loss with more data each than more incidents with less data." [6]

PERSES is also a good fit for large storage systems where a small but constantly shifting subset of the data is in active use. Systems that organically grow around a dynamic set of requirements naturally tend to have a small set of data in active use and a long tail of accesses across the remainder of the data [7]. These systems resemble archives in that while there are distinct, consistent projects, there is little repeat access for popularity based allocation to take advantage of. We study such archival-like workloads in this paper to form a baseline for improvement.

PERSES improves availability by laying out project groups adjacently on disk to isolate faults. Maintaining an out-of-band index of all of the data in a project requires significant administrative overhead. To avoid this, PERSES dynamically calculates associations in historical trace data to identify likely project groups. Since elements of a project are co-located on disk, any single failure impacts a very small number of projects. We derive project groups from either trace metadata

1526-7539/14 $31.00 © 2014 IEEE
DOI 10.1109/MASCOTS.2014.17

or access patterns. We use a statistical method to calculate projects in O(n), where n is the number of accesses in the training set, without the overhead of maintaining groups by hand.

To demonstrate PERSES, we injected failures into two dataset groupings and ran them through a disk simulator with a variety of layouts based on calculated project groups. We show that layout under PERSES leads to faults that affect fewer projects during the rebuild phase than a group-agnostic layout. Our fault injection simulation compares the real time lost during disk rebuilds across layouts by measuring *project hours gained*, which we define as the delay avoided by all project groups as a result of the access requests that are delayed while disks are rebuilding compared to the reference allocation.

We found that in our best case, PERSES gained over 4000 hours of project time during the three year trace compared to a random allocation and gained over 1000 hours compared to an ideal temporal allocation. Additionally, we found that increasing the IOPs did not significantly hurt the fault isolation. Finally, we discovered that restricting the minimum group size, thereby reducing grouping noise, improved the performance under PERSES even on smaller disks, with a maximum productivity improvement of 94% across large groups. Our main contributions are:

1. A methodology for project detection for better performance during failure events (PERSES).
2. A fault simulator with real trace data showing up to 80% decrease in project time lost with PERSES allocation.
3. A direct comparison of statistical and categorical project detection techniques.
4. Parameter identification and optimization strategies for rebuild speed and minimum group size.

II. BACKGROUND AND RELATED WORK

Managing availability in RAID-like systems has received attention commensurate with increasing disk size and corresponding rebuild difficulty [3]. Disk sizes have grown, but the read speed is limited by power concerns and by the areal density of data on the platters [8]. Additionally, online-reconstruction is increasingly becoming CPU-bound, meaning that the cost of on-line reconstruction is unlikely to go down any time soon [4].

Localizing data for fault isolation was first proposed by the D-GRAID project at the University of Wisconsin, Madison [9]. D-GRAID proposed to analyze the reliability increase of distributing blocks of a file across as many disks as possible versus keeping the blocks together on not only the same disk, but the same track. They proposed that since a head error is likely to affect adjacent track members more than random blocks on a disk, writing a file consecutively was an effective way to localize failures and thus minimize the project time lost as a result of file unavailability during the rebuild process. Our grouping methodology will allow for failures to be localized to projects, which represent working sets, allowing more of the system to be usable in case of failure.

Many large scale systems are provisioned to hold data for long, potentially archival periods of time. As systems scale, the concept of a "long" time to store data shortens. For instance, an exabyte scale storage system with 99.999% annual data retention will lose data in a decade comparable to what a petabyte scale storage system with equal retention loses in a century. While the percent remains constant, the value of data does not necessarily decrease as more of it is stored. Therefore, the point at which advanced reliability measures must be taken to ensure availability in a large storage system is likely to be sooner than we are accustomed to. Several studies have shown that storing data reliably over the archival time scale presents additional challenges, such as accumulated latent sector errors and data migrations, to the already difficult field of storage reliability [10]–[13]. A challenge of this size requires combining known techniques of enhancing reliability with methods optimized for the expected workload and time scale of archival storage.

Recent work has shown failures remain an issue for large disk-based storage systems. For example, Schroeder and Gibson showed that the average disk replacement rates in the field are typically between 2 and 4%, implying a high reconstruction rate [1]. Other studies have shown that latent sector errors can lead to drive failure [2]. Pinheiro *et al.* showed that failures are both difficult to predict and common [14]. Scrubbing techniques can efficiently catch these otherwise unobserved errors so that the disks can be rebuilt promptly, but it is still often necessary to rebuild much of the disk [15]. Modern storage systems can use parity to serve requests for data on a failed disk while the disk is re-building [16]. Many reconstruction optimizations have been proposed for distributed RAID rebuild [17], [18]. On-line reconstruction, serving accesses by reconstructing data on demand, has been shown to be between 3 and 70 times slower than serving requests from disk because of disk thrashing [19].

WorkOut addresses rebuild degradation by employing a surrogate RAID array to serve requests for "popular" data, where popular is defined as accessed twice within the course of the rebuild [19]. Tiair *et al.* also base their reconstruction algorithm on popularity [20]. These approaches are limited to workloads that have many repeat accesses to the same data in a very short period of time. In PERSES, we sidestep this limitation by exploiting correlation along multiple dimensions in addition to recency and frequency. Other papers such as Thomasian *et al.* have shown significant improvement in performance during rebuild by modifying the underlying parity structure or scheduling [20]–[22]. PERSES could be combined with existing optimizations for reconstruction to evaluate the combined impact, but it is likely that since PERSES is primarily a data allocation scheme, it can be combined with all of these techniques to further reduce the impact of failure events.

High-performance storage systems such as Ceph and GPFS that add reliability through mirroring are designed for systems where availability and performance trump costs [23], [24]. For a long term system, keeping several times the number of disks you have for data on hand is infeasible from a cost perspective, but many still need reasonable availability. Efforts have been made to use different erasure coded structures, but these still distribute failures more evenly than we believe is optimal for availability [25]–[28]. Disk-based systems such as Oceanstore [29] and SafeStore [30] combine mirroring, erasure codes, and strategic data placement to add reliability to their storage networks. While these systems are fast and highly

available, they are not optimized to be low power and low cost. Other tiered reliability schemes similarly lack an emphasis on cost and power management [31], [32].

Data layout is a major area of storage and filesystems research. Lamehamedi *et al.* look at the cost and reliability tradeoff of adding replicas, which we use to inform our later calculations about the benefit of storing multiple copies of records that are members of multiple projects [33]. Sun *et al.* show that for parallel systems, there is a strong argument for application-driven data placement [34]. However, none of these projects focus on layout for reconstruction.

Previous work in grouping data for power management indicated that in some long-term storage workloads, accesses happen in close temporal proximity within statistically separable groupings [35]. There is also evidence that these groups are predictive and can correspond to real-life working sets for applications, users, or projects [36]. A data layout designed for fault isolation is a natural progression from a power-aware data layout since both exploit highly correlated access patterns. Finally, many workloads show an inverse Pareto distribution for project access probabilities over time [36]. Under PERSES, failures that impact long tail of low access probability projects have a chance of having minimal to no effect on performance.

III. DESIGN

PERSES is a statistical layout algorithm designed to isolate faults to as few projects as possible. The two main functions of PERSES are project detection and layout. Projects are automatically learned by extrapolating relationships based on a period of accesses and modifying these groups as new data enters the system. This unsupervised approach to project detection is a better fit for dynamic data where the meaning of curated labels drifts over time, or for systems with privacy controls or performance barriers to collecting metadata-rich traces. Since we learn our projects, they may not map precisely to real applications or users. In fact, to avoid overfitting to past data, it is important that the mapping be inexact.

In order to lay out data by project, we need to know what the projects are. Manually maintaining projects has high administrative overhead and presents a consistency problem [37]. To avoid these issues, we automatically identify sets of data that have a high likelihood of being accessed within a short time of each other and label these as *projects*. We extract these projects by collecting disk accesses and reasoning about the patterns in the trace, either using metadata or statistical analysis. In either case, projects are determined by analyzing a portion of the accesses and are then applied to the remainder. Groups are re-calculated at intervals determined by a running average of predictive power for the current grouping.

We started with the hypothesis that the particular grouping technique used is unimportant as long as elements within the same project have a high probability of co-access. We tested two statistical groupings against a categorical labeling for an archival dataset.

A. Project Detection with NNP

For the statistical analysis, we use a variant of N-Neighborhood Partitioning (NNP) [36], a grouping algorithm that accumulates a window of I/Os, calculates a sparse $n \times n$ distance matrix over the window, determines projects based on density, and then updates the prior grouping based on this new information. We chose NNP because it biases towards smaller groups, reducing the possibility of cache churn, and because it runs in $O(n)$, which allows us to quickly reclassify when predictivity begins to drop.

The first step is to select a temporal window of accesses with size w. We select $w = 500,000$ accesses, corresponding to about 250 MB of the access trace, because of local memory constraints. NNP requires up to $O(w^2)$ to calculate a pairwise distance matrix between elements in the window. However, this matrix is typically very sparse since we only consider similarity above a threshold. Our implementation groups these accesses in under 10 minutes in a Python simulator on a quad-core machine with 16GB of memory. A larger window can detect groupings that contain more elements and also have stronger intra-group similarity, but increasing the size of the window quickly meets diminishing returns [37]. The windows overlap by twice the current average project size to limit over counting.

For each window, the partitioning steps are:

1. Calculate the pairwise distance matrix
2. Calculate the neighborhood threshold and detect projects in the I/O stream
3. Combine the new grouping with any prior groupings

Each access in the window is represented as a timestamp, t, and unique identifier, o. The identifier o corresponds to current spatial location. For n accesses in a window, we represent pairwise distance between every pair of accesses (p_i, p_j), as an $n \times n$ matrix d with $d(p_i, p_i) = 0$. We calculate the distances in this matrix using weighted Euclidean distance where a point $p_i = (t_i, o_i)$.

We were most interested in recurring identifier pairs that were accessed in short succession. As a result, we also calculated an $m \times m$ matrix, where m is the number of unique identifiers in our window. This matrix was calculated by identifying all the differences in timestamps, $T = \{t_{ik} - t_{jl} \; \forall \, k \in I, l \in J\} : T_1 = [T_1 = t_{i1} - t_{j1}, T_2 = t_{i1} - t_{j2}, T_3 = t_{i2} - t_{j1}, \ldots]$, between the two identifiers o_i and o_j, where I is all instances of o_i in the trace and J is all instances of o_j. To avoid overfitting, we treat the unweighted average of these timestamp distances as the time element in our distance calculation. Thus, the distance between two identifiers is:

$$d(o_i, o_j) = \sqrt{\left(\frac{\sum_{i=1}^{|T|} T_i}{|T|}\right)^2 + s \times (o_i - o_j)^2}$$

We combine the temporal distance with the spatial distance between data, which is a relatively weak but not insignificant predictor. Here, s is a scaling factor based on the typical relative distance between identifiers versus time. Once the distance matrix is calculated, we calculate a value for the neighborhood threshold, $\check{N}$. In the online case, $\check{N}$ must be selected *a priori* and then re-calculated once enough data has entered the system to smooth out any cyclic spikes. Once the threshold is calculated, the algorithm looks at every access in turn. The first access starts as a member of group g_1. If the next access occurs within $\check{N}$, the next access is placed into

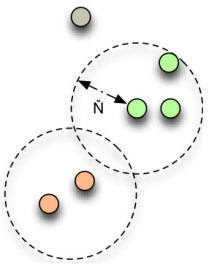


Fig. 1. Each incoming access is compared to the preceding access to determine whether it falls within the neighborhood ($\hat{N}$) to be in the same group. If it does not, a new group is formed with the incoming access.

group g_1, otherwise, it is placed into a new group g_2, and so on. Figure 1 illustrates a simple case.

1) Combining Neighborhood Partitions: A grouping G_i is a set of projects $g_1, \ldots, g_w$ that were calculated from the i^{th} window of accesses. Unlike standard neighborhood partitioning, NNP is not entirely memoryless; NNP combines groupings from newer data to form an aggregate grouping. We do this through fuzzy set intersection between groupings and symmetric difference between projects within the groupings. So, for groupings $G_1, G_2, \ldots G_z$, the total grouping G is :

$$G = (G_i \cap G_j) \cup (G_i \Delta_g G_j) \quad \forall i, j \quad 1 \leq i, j \leq z$$

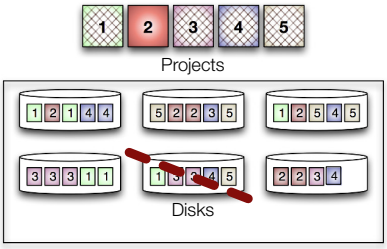
where Δ_g, the groupwise symmetric difference, is defined as every group that is not in $G_i \cap G_j$ and also shares no members with a group in $G_i \cap G_j$. For example, for two group lists $G_1 = [(x_1, x_4, x_7), (x_1, x_5), (x_8, x_7)]$ and $G_2 = [(x_1, x_3, x_7), (x_1, x_5), (x_2, x_9)]$, the resulting grouping would be $G_1 \cap G_2 = (x_1, x_5) \cup G_1 \Delta_g G_2 = (x_2, x_9)$, yielding a grouping of $[(x_1, x_5), (x_2, x_9)]$. (x_1, x_4, x_7), (x_1, x_3, x_7), and (x_8, x_7) were excluded because they share some members (*e.g.*, x_7) but not all. This group calculation happens in the background during periods of low activity. As accesses come in, we need to update projects to reflect a changing reality. We do this by storing a likelihood value for every group. This numerical value starts as the normalized median intergroup distance value and is incremented when two elements of a group are accessed within 50s of one another [35].

NNP is especially well suited to rapidly changing usage patterns because individual regions do not share information until the group combination stage. When an offset occurs again in the trace, it is evaluated again, with no memory of the previous occurrence. Combining the regions into a single grouping helps mitigate the disadvantage of losing the information of repeated correlations between accesses without additional bias.

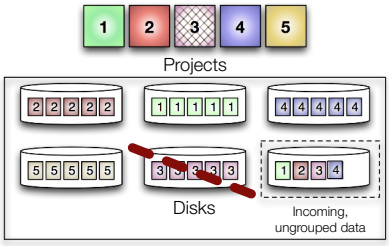
2) Runtime: Neighborhood partitioning runs in $O(n)$ for n accesses since it only needs to pass through each neighborhood twice: once to calculate the neighborhood threshold and again to collect the projects. This makes it an attractive grouping mechanism for workloads with high IOPS, where a full $O(n^2)$ comparison is prohibitive. Additionally, NNP captures projects in real time and lets PERSES quickly take advantage of shifts in access patterns.

B. Layout

Most storage systems spread out data across several disks to maximize resilience against disk failure. When a disk fails,



(a) Standard. Single failure impacts Projects 1,3,4,and 5



(b) With PERSES. Single failure impacts Project 3

Fig. 2. Projects are represented by numbered boxes. When a disk fails, all projects that have data on the disk have degraded performance. PERSES dynamically groups data together on disk to limit the performance impact of failures across projects. Note that this is independent of redundancy arrangements.

access to data on the failed disk is degraded from having to reconstruct data from parity or backups. PERSES lays out data according to project to reduce the impact these periods of degraded access have on the users of the system. The key insight is that the impact on productivity for a project can be disproportionate to the amount of data lost. For example, missing an input file to a scientific simulation can cause the simulation to stall and overrun its allocated time. Missing part of a virtual machine boot sequence can cause several users to have to wait to restore images. Missing a single small file could break the build for an entire code base.

Our layout places as much project data as possible on the same disk. This way, when a failure occurs it impacts fewer projects, which means the system is more productive. The ideal disk failure is caught by a monitoring process (such as disk scrubbing) and repaired before any accesses are made to the lost data. We refer to this as a *zero-impact* failure. Intuitively, on systems with bursty access patterns, zero-impact failures become increasingly common when data has high locality. One high level goal for PERSES is to maximize the number of failures that do not impact system performance. This does have potential load balancing implications, which we discuss further in Section VI-A.

Figure 2 shows what a failure would look like on a trivial shared system with and without PERSES. The numbered boxes correspond to blocks of data, and the numbers correspond to projects. These projects could be sets of user data, large data set analyses on enterprise, scientific experiments, code bases, etc. In the ungrouped example, Figure 2(a), data is arranged without any consideration to project membership. When one disk is lost, indicated by the thick dashed line, up to four projects could see performance degradation if they are accessed before rebuild completes. On a system laid out with

PERSES, such as Figure 2(b), the data is laid out across the disks based on project membership. When a disk fails in this scenario, only Project 3 is affected since it is the only project on the failed disk, both reducing the amount of degradation across groups and increasing the probability that the failure is zero-impact. Groupings are refreshed periodically and the layout re-arranged, so some data will always be unlabeled. This unlabeled data is written serially to a reserved area on disk until the next grouping cycle occurs.

We assume that the underlying system uses parity to rebuild data when a disk is lost, but we do not specify a particular reliability scheme. This is intentional since even though PERSES is designed for a parity-based reliability architecture, all reliability schemes save for local mirroring introduce a reconstruction delay. We care about reconstruction speed and consequent time spent degraded, but not the reliability method. Rebuild requests are typically given very low priority compared to incoming reads and writes, so a parity system or backup server should have little overhead that we do not account for [16]. In our simulation, we model a range of rebuild speeds to better understand the impact of PERSES in different reliability environments.

PERSES is designed for a system with multiple disks, different projects that rely on many blocks of data, and a controller to group data and manage layout. We also assume the ability to collect minimal trace data to inform the grouping algorithm. The best candidate systems for PERSES have a number of data disks greater than or equal to the number of projects or working set groups on the system, though we see in Section V that this is not strictly necessary.

IV. EXPERIMENTS

We built a trace simulator with stochastic fault injection to analyze PERSES on a range of different hardware configurations and rebuild environments.

Our simulator goes through the following steps:

1. Initialize disks and cache
2. Determine initial groupings
3. Lay out data across disks
4. Step through a real data trace; update grouping as needed

Disks are initialized with all of the data that is read through the course of the trace. Disks are then filled progressively with either grouped data, data placed in order of trace appearance, or randomly placed data such that the only empty space is on the final disk, depending on the experiment. The amount of data each trace accesses is fixed, so as we add more disks to the simulator there is less data per disk. Since we do not know the size of the accesses in our trace, we arbitrarily allocate 10 MB per access, resulting in a total data size of 880 GB and 50 TB for our datasets. We retain generality since our simulation results can be translated to any system with fixed size blocks by adjusting the data to disk ratio accordingly. The simulator uses an LRU read cache to capture popular accesses. We chose LRU because it is well understood, and we did not explore further because, for our workloads, cache size had no impact on our results until the cache was over 100 GB. The default cache in the following experiments starts cold and is

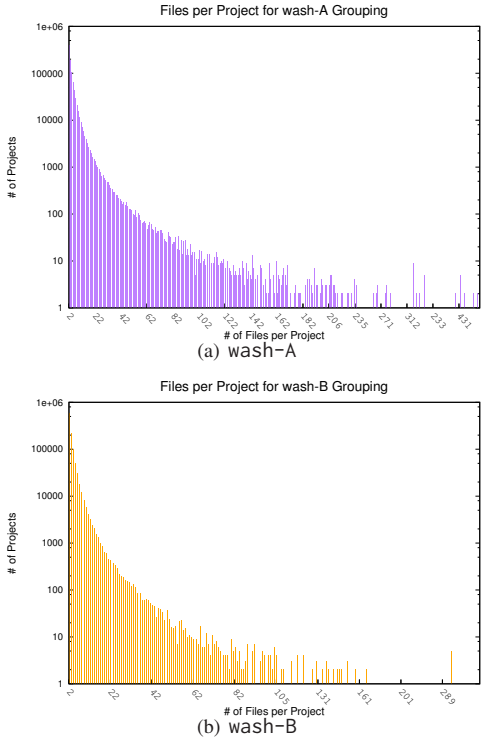


Fig. 3. Project size distributions for statistical groupings. Both wash-A and wash-B statistical groupings have enough very small projects to greatly skew the project size distribution towards larger projects. Groups sizes obey an inverse Pareto distribution.

10 GB. This cache is assumed to be in memory and unaffected by disk failure.

An initial grouping is calculated on training data before the trace is run, as described in Section III-A. Groups are then laid out sequentially on disks starting with the smallest projects. In the ungrouped experiments, records are laid out randomly or temporally without attention to project membership. Modeling correlated failure and exploring alternate strategies for assigning projects to disks are in our future work.

Disks are initialized with a uniform failure probability of 1 in 10^{-5} per time step. After each access, the probability of failure is increased by 1 in 10^{-7} to represent wear on the device, which we base on a study by Pinheiro *et al.* [38]. Only full-disk failures are considered since recent work has shown latent sector errors to be surprisingly rare on modern hardware [39].

We express disk rebuild speed with a single parameter, r, that encompasses the disk bandwidth, network overhead, and CPU load of a disk to form the number of seconds it takes to restore a gigabyte of data. We choose $r = 30$ s/GB (≈ 34 MB/s) as our default value for reconstructing data based on the 50 MB/s read speed of an off-the-shelf 7200 RPM disk [4]. This is a low estimate since CPU saturation and not read speed is the typical bottleneck for disk rebuild [4]. We also test on r values as low as 10 s/GB ($\approx$102 MB/s) to demonstrate that PERSES can improve layout even on advanced hardware.

We define *project time lost* as the total delay encountered by all projects as a result of the access requests that are delayed

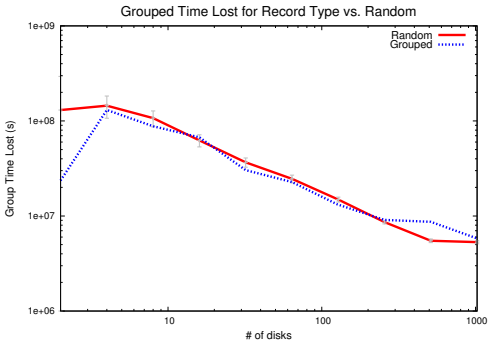


Fig. 4. Categorical: 1060 hours gained; $r = 30$s/GB

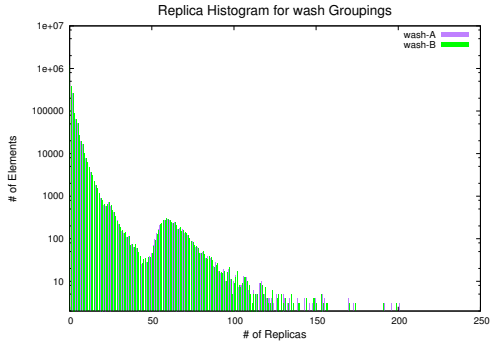


Fig. 6. Elements overwhelmingly appear in only one group, but can appear in as many as 300. Note that this includes groups that have low likelihood.

while disks are rebuilding. The total project time lost can be higher than the total time that the system is in degraded mode since a failure, especially on randomly allocated disks, can affect multiple projects. Once the data is laid out on disk, the trace is played in the simulator to calculate the total project time lost as a result of failure events. While failures are random, they are set to a consistent random seed between each pair of grouped and random runs to make the runs comparable. In experiments with fewer disks, our results had very high variance regardless of the number of trials because well timed failures can cascade. All random layout runs were run at least 50 and typically over 100 times. One benefit of the PERSES layout in real systems may be the relative predictability of time lost compared to random allocation.

A. Data

We use two statistical groupings and one categorical grouping derived from a corpus of accesses to a database of vital records from the Washington state digital archives (`wash`) in this work. We chose these traces because they had rich metadata, so we could for the first time provide a direct comparison between statistically and categorically defined projects.

Accesses in `wash` are labeled with one of many type identifiers (*e.g.* "Birth Records," "Marriage Records") that we use for categorical grouping [40]. We examined $5,321,692$ accesses from 2007 through 2010 that were made to a 16.5 TB database. In addition to the supplied type identifiers, each Record has a RecordID that is assigned as it is added to the system. We use these RecordIDs as a spatial dimension when calculating statistical groupings as discussed in Section III-A.

V. SIMULATION RESULTS

We ran our fault simulator on categorically grouped `wash` and statistically grouped `wash-A` and `wash-B` datasets. Surprisingly, the categorical grouping underperformed both statistical groupings. We measure the impact of our allocation by averaging the amount of time PERSES gains versus the layout it is being compared against when it has more than two disks over the course of the trace; we call this number *hours gained*.

A. Categorical Groupings

Figure 4 shows that categorical grouping somewhat reduces the project time lost compared to random allocation for $r = 30$ s/GB. Additionally, we found that increasing r to 100 s/GB. had negligible effect on the relative project hours gained.

B. Statistical Groupings

NNP parameters that affect statistical grouping include s along with weights for means and standard deviations in the partitioning calculation. Lacking any external confirmation of validity, we fully explored the parameter space and then calculated the average project size of all of the resultant groupings.

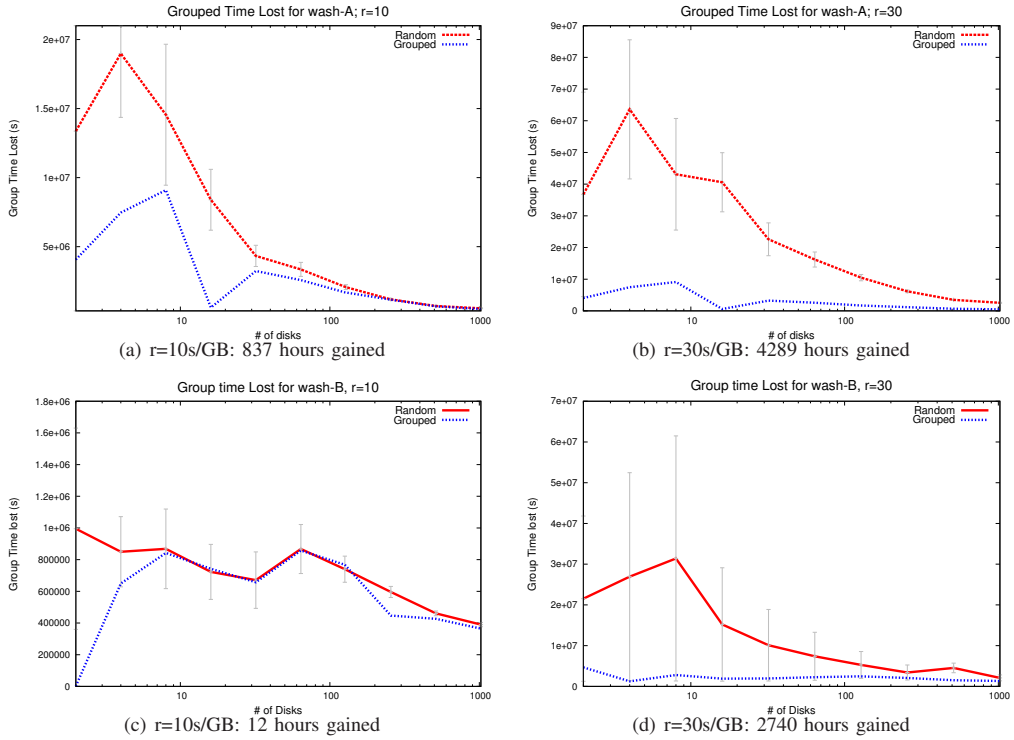
TABLE I. NNP GROUPING STATISTICS: `wash`

	Avg. Group Size	Std. Dev.	Max. Group Size
`wash-A`	4.7	8.3	1865
`wash-B`	3.2	4.0	1012

When these groupings were clustered using the parameters and average project size as features, elements fell into one of three clusters of almost identical groupings. The first cluster, which resulted from extreme parameter combinations, was the "null" grouping where every element is a separate group. We name the representatives we selected from the two non-trivial grouping clusters `wash-A` and `wash-B` . Table I shows the main differences between the two groupings. Though the difference in average project sizes seems small, the project size distribution within the grouping (Figure 3) shows that the inverse Pareto distribution of project sizes results in many more larger projects for the `wash-A` grouping.

Figures 5(a) and 5(b) show how allocating data on disk using the `wash-A` grouping improves project time lost by up to 50% at 4 disks going down to approximately 5% as the number of disks increases. This represents an increase of an order of magnitude for larger disks with r, the number of seconds it takes to rebuild a gigabyte of data, as low as 10. At $r = 30$ s/GB, there is a clear benefit to laying out disks with the `wash-A` grouping with the percent improvement ranging from 10% for small disks to 80% for larger disks. The `wash-B` grouping, on the other hand, did not consistently outperform random allocation (Figures 5(c) and 5(d)). This is because `wash-B` has smaller, noisier projects, which biases the layout algorithm towards placing related data farther apart on average. We discuss this in greater depth in Section VI.

1) Re-Replication: We designed PERSES to only store information in the most likely project. In a real implementation, we need to simultaneously optimize for both availability and bandwidth. To improve bandwidth across the system, one could replicate records that are members of multiple projects. Figure 6 shows the total number of projects records appear



(a) r=10s/GB: 837 hours gained

(b) r=30s/GB: 4289 hours gained

(c) r=10s/GB: 12 hours gained

(d) r=30s/GB: 2740 hours gained

Fig. 5. PERSES significantly reduces project time lost with the `wash-A` grouping and less so with the `wash-B` grouping.

in, including projects that are subsets of other projects. The average number of projects a record is a member of is 2.48. Replicating data on a system, in groups or otherwise, significantly improves the reliability of the system as well as distributing the data to avoid network level points of throttling [41].

C. Extensions

To better understand PERSES, we explored the effects of different control layouts and rebuild speeds along with limiting group size and increasing the I/Os per second (IOPS) in our trace. The minimum group size experiments came from the observation that larger groups had more to lose and potentially experienced more benefit from localization. We found that restricting group size did improve the performance of PERSES. Finally, we realize that our traces have relatively low IOPS, and we wanted to address concerns about the efficacy of PERSES in a more active environment. To do this, we speed up our trace by a factor of ten and show that PERSES improves project time lost even on the compressed trace.

Temporal Layout

We compare our project based layout against a random layout since we have no information as to the actual arrangement of the data behind our traces. To provide a more challenging comparison, we also generated a "temporal" layout where records are placed on disk by an oracle in the order the records will later be read in the trace. This arrangement represents the best case layout possible for this data in a situation with complete fore-knowledge, and thus serves as an upper goal for our method. Figure 7 shows that even compared to this ideal layout, PERSES at worst matches the project time lost for all but the largest disks, with an average of 1300 hours of project time saved over the three years of the trace.

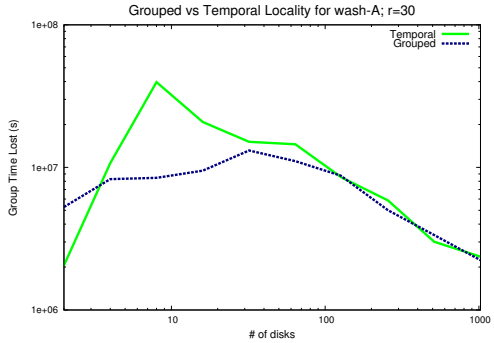
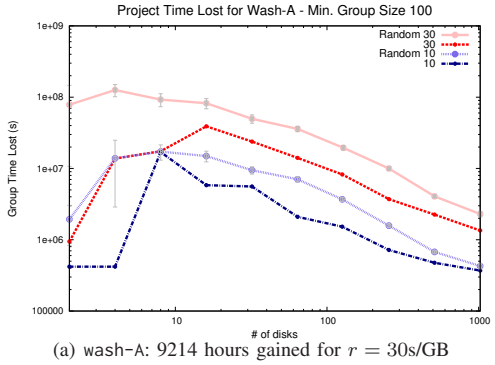


Fig. 7. PERSES performs as well as or better than data allocated with a temporal oracle. 1321 hours gained.

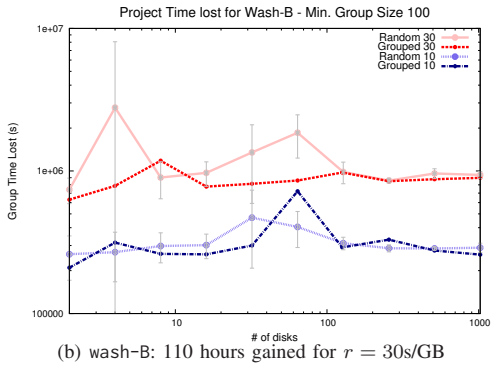
Minimum Group Size

We hypothesized that `wash-A` outperformed `wash-B` because `wash-A` has a higher average group size, and proceeded to test the effect of raising the minimum group size used in PERSES layout.

Figure 8 shows both `wash-A` and `wash-B` groupings with all project groups of fewer than either 50 or 100 members removed. The top pair of lines on each graph correspond to a rebuild rate of $r = 30$ s/GB and the bottom pair correspond to $r = 10$ s/GB. Surprisingly, if we restrict the size of groups



(a) wash-A: 9214 hours gained for $r = 30$s/GB



(b) wash-B: 110 hours gained for $r = 30$s/GB

Fig. 8. wash-A saves almost three times as many hours when the group size is restricted to 100 elements, but wash-B suffers a significant penalty. Time lost is on a log scale.

to only model larger projects, we see significant improvement even with $r = 10$ s/GB, which corresponds to a rebuild rate of 102 MB/s, with the wash-A grouping. Equally surprising, the wash-B grouping suffers a significant decline in performance.

This leads to two insights. First, wash-A outperforms wash-B because wash-A has more large groups: with restricted group size, wash-B does not have enough internal structure and information to outperform random. Secondly, the performance of wash-A even at $r = 10$ s/GB indicates that with a grouping which biases towards larger groups, PERSES is valuable even on high-end hardware with very fast rebuild. Our results indicate that PERSES has higher impact on larger groups since without group-based allocation a project is more likely to be spread across many disks.

High IOPS

Finally, we had some concerns about how PERSES would behave under high IOPS. To test this, we compressed our traces by a factor of 10, such that for example accesses that are 100 s apart in the trace are 10 s apart in the compressed trace. Figure 9 shows that while PERSES does not save as much project time lost under compression, it still handily outperforms random allocation once disks are reasonably sized.

VI. DISCUSSION

Our results show that for most cases, PERSES significantly reduces the time lost across projects in our two workloads.

The most surprising result of our simulations is that statistical grouping significantly outperformed categorical. This

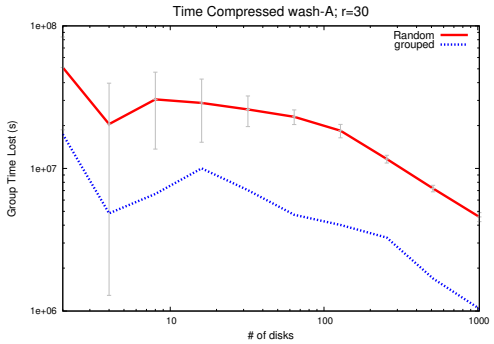


Fig. 9. Compressing the accesses in wash-A by a factor of 10 only slightly hurts the performance of PERSES

is because, given the relatively short rebuild period, PERSES favors groupings with high co-access probability. Categorical groupings do not take into account the actual usage of group members, instead only considering semantic similarity. On the other hand, high probability of co-access defines the statistical groups produced by NNP. Over a longer trace, we expect that the categorical groups will perform better, but not overcome statistical groupings as long as there are few sudden and drastic shifts in usage.

The statistical grouping results help illustrate why PERSES works. wash-A strictly outperforms wash-B because wash-B is strongly biased towards small group sizes. In the wash-B grouping, the majority of records are members of small projects. These projects, clustered together on relatively few disks, each contribute equally to project time lost on drive failure, inflating the time lost number relative to the wash-A grouping. A layout algorithm optimized to distribute small groups across disks could potentially alleviate this issue.

Simply restricting the group size is insufficient because if the grouping biases towards small groups, removing those groups leaves the majority of records without a project association. This is why wash-B performs only slightly better than random in the case where the minimum group size is 100 records. wash-A performs especially well in the restricted group size case because it biases towards larger groupings, so removing smaller groups removes weak groups more than it adds information. In larger systems, the layout overhead may become more significant as the workload changes.

The bottleneck in rebuilding data is the speed a disk can be read. The rebuild rate for most of our experiments, $r = 30$ s/GB, corresponds to the 50 MB/s high end disks available today. PERSES will perform even better on slower disks, since each disk failure leads to higher project time lost per project, amplifying the effects of locality. From another perspective, the probability that the failure will be observed by an access increases with downtime.

A. Redundancy and Load Balancing

One concern with grouping everything a project needs on a small set of physical devices is a slightly higher probability of losing all of the data for a project in a data loss event. There are two ways to mitigate this worry of "putting all ones eggs in the same basket." Though we do not assume a particular reliability method, PERSES is designed for a system

with parity groups, and can easily integrate into existing parity schemes. These groups can be arranged on a segment level instead of a disk level, distributing the risk. Additionally, our grouping method replicates data that is in multiple groups. This, by definition, replicates data that is useful to multiple projects, making exactly that data with the greatest net value the most replicated and thus most robust. Finally, any tertiary storage, such as backups, should be arranged in a project-agnostic layout to provide additional resilience against data loss.

Another issue with grouping all of the data for a project on a small number of devices is that repeated, frequent accesses to the same disk could impact performance while the project is in use. One clear way to address this issue is to place entire groups into a group-aware LRU cache when a member of the group is accessed. All successive accesses to a group are made from memory. In this case, we are not trying to maximize accesses to the cache as much as minimize repeated accesses to a device, so a simple cache policy suffices. Another major concern is the performance impact of rearranging groups on disk when groups are recalculated. Since the grouping loses predictive power slowly over time, groups can be rearranged lazily as disks are used or during low request periods.

B. Interaction with Existing Technologies and Performance

Many strategies exist to improve storage performance during rebuild. PERSES addresses layout, amplifying the benefit of current pre-fetching techniques by reducing disk overhead. We cannot outperform strategies that specialize the layout using external knowledge of the expected workload characteristics; our techniques are designed to be easily adaptable to different, unknown workload types. As more services move to massive multi-project storage systems such as S3, optimization strategies that do not rely on domain knowledge will be essential. In these systems, if user requests are slow enough to time out, the provider will break the SLA, so PERSES applied to the user data has the potential to save cloud providers money and reputation by isolating degradation to few users.

Rapid rebuild is touted as making RAID reconstruction irrelevant, but the latent sector errors it fixes only happen in rapid succession, meaning that it will always be playing catch-up [1]. Secondly, we see improvement with PERSES even at $r = 10$, which corresponds to reconstructing data from a failed drive at over 100 MB/s, faster than most drives can even be read. Finally, we also experimented with modifying the read cache size and found that it did not significantly alter our results. This indicates that layout models based on the popularity of data for placement would have performed poorly on our traces.

Based on the results from increasing the IOPS of our data, we can extrapolate our results to workloads with more activity such as enterprise or HPC. As IOPS increases, PERSES still saves several hundred hours in project time over three years. In a real high IOPS trace, this number would likely be higher because instead of projects being accessed faster, more projects are being accessed. Thus, the time lost should be closer to the normal case than the high IOPS case we simulate here. Finding data sets with labels for categorical grouping is very challenging. While our current results are compelling, we are actively seeking other workload types to test.

Performance under PERSES is much more predictable than under arbitrary layouts, which could allow system administrators to set better SLAs and predict patterns in their workflows. Regrouping runs in $O(n)$, and regrouped data can be moved lazily [36]. PERSES is general purpose and requires no administrative overhead to find groups or re-arrange data, and it can combine with existing rapid reconstruction methods to help alleviate the performance impact of disk rebuild, and may help RAID-like systems scale to match the demands of the cloud.

C. Future Work

Our next steps are to add correlated failures to our fault simulator to represent more failure scenarios. We expect PERSES to do well with correlated failures because there will be fewer, larger failures, which PERSES is designed for. Another interesting problem is in how projects are allocated to disks. Currently, disks are filled in with projects based on project size. We are exploring using more intelligent bin packing to place projects on disk based on probability of access. Finally, we are looking at the effects of combining PERSES with existing systems to reduce reconstruction overhead through replication and caching.

Our eventual goal is to design a data layout algorithm for non-hierarchical file systems. Current file systems use the directory hierarchy to obtain some notion of likelihood of co-access in data. If we can automatically detect projects and lay them out such that they are are isolated, we can control fragmentation in non-hierarchical systems without administrative overhead or time consuming metadata analysis.

VII. CONCLUSIONS

We have demonstrated PERSES, a data layout technique that significantly reduces the impact of system degradation in large multi-use storage systems. We have shown that we can automatically derive projects from statistical data and apply them to lay out data such that a failed disk affects few projects, and these statistical projects outperform projects derived from metadata. Our experiments show that we can reduce the total time that projects perceive as lost by up to 80%, which corresponds to over 4000 hours over three years. We also showed that even against an ideal temporal layout, PERSES saves over 1300 hours in our trace. Furthermore, PERSES is especially effective using groupings biased towards larger groups. Finally, we showed that PERSES can operate with high IOPS, making it relevant for active systems including enterprise and cloud storage.

ACKNOWLEDGMENTS

This research was supported in part by the NSF under awards CNS-0917396 (part of the American Recovery and Reinvestment Act of 2009 [Public Law 111-5]) and IIP-0934401 and by the DOE under Award Number DE-FC02-10ER26017/DE-SC0005417 We also thank CRSS sponsors and all SSRC members for their generous support.

REFERENCES

[1] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?" in *FAST'07*, Feb. 2007, pp. 1–16. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/schroeder-fast07.pdf

[2] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler, "An analysis of latent sector errors in disk drives," in *Proceedings of the 2007 SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, Jun. 2007. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/bairavasundaram-sigmetrics07.pdf

[3] M. Kryder and C. Kim, "After hard drives - what comes next?" *Magnetics, IEEE Transactions on*, vol. 45, no. 10, pp. 3406–3413, 2009.

[4] Louwrentius, "Raid array size and rebuild speed," 2010. [Online]. Available: http://louwrentius.com/blog/2010/04/raid-array-size-and-rebuild-speed/

[5] F. Gens, "Idc predictions 2013: Competing on the 3rd platform," *IDC, Report, Dec*, 2013.

[6] A. Cidon, S. M. Rumble, R. Stutsman, S. Katti, J. Ousterhout, and M. Rosenblum, "Copysets: Reducing the frequency of data loss in cloud storage," in *USENIX ATC'13*, 2013, pp. 37–48.

[7] C. Staelin and H. Garcia-Molina, "Clustering active disk data to improve disk performance," *Princeton, NJ, USA, Tech. Rep. CS–TR–298–90*, 1990.

[8] R. Wood, "Future hard disk drive systems," *Journal of magnetism and magnetic materials*, vol. 321, no. 6, pp. 555–561, 2009.

[9] M. Sivathanu, V. Prabhakaran, A. Arpaci-Dusseau, and R. Arpaci-Dusseau, "Improving storage system availability with D-GRAID," *ACM TOS*, vol. 1, no. 2, pp. 133–170, 2005.

[10] B. Schroeder and G. Gibson, "Understanding failures in petascale computers," in *Journal of Physics: Conference Series*, vol. 78. IOP Publishing, 2007, p. 012022.

[11] M. Baker, M. Shah, D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, T. Giuli, and P. Bungale, "A fresh look at the reliability of long-term digital storage," in *Proceedings of EuroSys 2006*, Apr. 2006, pp. 221–234. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/baker-eurosys06.pdf

[12] P. Constantopoulos, M. Doerr, and M. Petraki, "Reliability modelling for long term digital preservation," in *9th DELOS Network of Excellence thematic workshop Digital Repositories: Interoperability and Common Services, Foundation for Research and Technology-Hellas (FORTH)*. Citeseer, 2005.

[13] S. Nath, H. Yu, P. B. Gibbons, and S. Seshan, "Subtleties in tolerating correlated failures in wide-area storage systems," in *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI)*, 2006. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/nath-nsdi06.pdf

[14] E. Pinheiro, W. Weber, and L. Barroso, "Failure trends in a large disk drive population," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST07)*, 2007.

[15] T. Schwarz, S. J. and E. L. Miller, "Store, forget, and check: Using algebraic signatures to check remotely administered storage," in *ICDCS '06*, Jul. 2006. [Online]. Available: http://www.ssrc.ucsc.edu/Papers/schwarz-icdcs06.pdf

[16] R. Hou, J. Menon, and Y. Patt, "Balancing i/o response time and disk rebuild time in a raid5 disk array," in *System Sciences*, 1993.

[17] Q. Xin, E. L. Miller, and T. J. E. Schwarz, "Evaluation of distributed recovery in large-scale storage systems," in *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, Honolulu, HI, Jun. 2004, pp. 172–181. [Online]. Available: http://www.ssrc.ucsc.edu/Papers/xin-hpdc04.pdf

[18] Q. Xin, E. L. Miller, T. J. Schwarz, D. D. E. Long, S. A. Brandt, and W. Litwin, "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. [Online]. Available: http://www.ssrc.ucsc.edu/Papers/xin-mss03.pdf

[19] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao, "Workout: I/o workload outsourcing for boosting raid reconstruction performance," in *FAST09*, 2009.

[20] L. Tiair, H. Jiang, D. Feng, H. Xin, and X. Shir, "Implementation and evaluation of a popularity-based reconstruction optimization algorithm in availability-oriented disk arrays," in *MSST '07*, 2007.

[21] L. Jones, M. Reid, M. Unangst, and B. Welch, "Panasas tiered parity architecture," *Panasas White Paper*, 2008.

[22] A. Thomasian, Y. Tang, and Y. Hu, "Hierarchical raid: Design, performance, reliability, and recovery," *Journal of Parallel and Distributed Computing*, 2012.

[23] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*. USENIX, Jan. 2002, pp. 231–244. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/schmuck-fast02.pdf

[24] S. Weil, S. Brandt, E. Miller, D. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in *OSDI 2006*. USENIX Association, 2006, p. 320.

[25] K. Greenan, E. Miller, T. Schwarz, and D. Long, "Disaster recovery codes: increasing reliability with large-stripe erasure correcting codes," in *Proceedings of the 2007 ACM workshop on Storage security and survivability*. ACM, 2007, pp. 31–36.

[26] M. Storer, K. Greenan, E. Miller, and K. Voruganti, "Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage," in *6th USENIX FAST*. USENIX Association, 2008, pp. 1–16.

[27] A. Wildani, T. Schwarz, E. Miller, and D. Long, "Protecting against rare event failures in archival systems," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems, 2009. MASCOTS'09. IEEE International Symposium on*. IEEE, 2009, pp. 1–11.

[28] Y. Saito, S. Frolund, A. Veitch, A. Merchant, and S. Spence, "FAB: Building distributed enterprise disk arrays from commodity components," in *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2004, pp. 48–58. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/saito-asplos04.pdf

[29] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells *et al.*, "Oceanstore: An architecture for global-scale persistent storage," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 5, pp. 190–201, 2000.

[30] R. Kotla, L. Alvisi, and M. Dahlin, "SafeStore: a durable and practical storage system," in *Proceedings of the 2007 USENIX Annual Technical Conference*, Jun. 2007, pp. 129–142. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/kotla-usenix07.pdf

[31] K. Gopinath, N. Muppalaneni, N. Kumar, and P. Risbood, "A 3-tier RAID storage system with RAID1, RAID5 and compressed RAID5 for Linux," in *2000 USENIX ATC*. USENIX Association, 2000, p. 30.

[32] N. Muppalaneni and K. Gopinath, "A multi-tier RAID storage system with RAID1 and RAID5," in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, 2000, pp. 663–671.

[33] H. Lamehamedi, Z. Shentu, B. Szymanski, and E. Deelman, "Simulation of dynamic data replication strategies in data grids," in *IPDPS'03*, 2003.

[34] X. Sun, Y. Chen, and Y. Yin, "Data layout optimization for petascale file systems," in *PDSW '09*. ACM, 2009.

[35] A. Wildani and E. Miller, "Semantic data placement for power management in archival storage," in *PDSW '10*. IEEE, 2010.

[36] A. Wildani, E. Miller, and L. Ward, "Efficiently identifying working sets in block i/o streams," in *SYSTOR '11*, 2011, p. 5.

[37] E. Pinheiro, W. Weber, and L. Barroso, "Failure trends in a large disk drive population," in *FAST'07*, 2007.

[38] B. Schroeder, S. Damouras, and P. Gill, "Understanding latent sector errors and how to protect against them," *ACM Transactions on Storage (TOS)*, vol. 6, no. 3, p. 9, 2010.

[39] I. Adams, M. Storer, and E. Miller, "Analysis of workload behavior in scientific and historical long-term data repositories," *ACM TOS*, vol. 8, no. 2, p. 6, 2012.

[40] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*. Bolton Landing, NY: ACM, Oct. 2003. [Online]. Available: http://www.ssrc.ucsc.edu/PaperArchive/ghemawat-sosp03.pdf