PERSES: Data Layout for Low Impact Failures

Technical Report UCSC-SSRC-12-06
September 2012

A. Wildani
avani@cs.ucsc.edu

E. L. Miller
elm@cs.ucsc.edu

I. F. Adams
iadams@cs.ucsc.edu

D. D. E. Long
darrell@cs.ucsc.edu

Storage Systems Research Center
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
http://www.ssrc.ucsc.edu/

PERSES: Data Layout for Low Impact Failures

Abstract

Disk failures remain common, and the speed of reconstruction has not kept up with the increasing size of disks. Thus, as drives become larger, systems are spending an increasing amount of time with one or more failed drives, potentially resulting in lower performance. However, if an application does not use data on the failed drives, the failure has negligible direct impact on that application and its users. We formalize this observation with PERSES, a data allocation scheme to reduce the performance impact of reconstruction after disk failure.

PERSES reduces the length of degradation from the reference frame of the user by clustering data on disks such that working sets are kept together as much as possible. During a device failure, this co-location reduces the number of impacted working sets. PERSES uses statistical properties of data accesses to automatically determine which data to co-locate, avoiding extra administrative overhead. Trace-driven simulations show that, with PERSES, we can reduce the time lost due to failure during a trace by up to 80%, or more than 4000 project hours over the course of three years.

1 Introduction

Over time, failure events are inevitable in large disk-based storage systems [4, 15]. 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 [8, 10]. 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 online. We look at this problem from the perspective of *working sets*: groups of files that are frequently accessed together, such as files for a project. If a failed disk contains data for multiple working sets or projects, all of those projects could stall until the rebuild is completed. If the failure occurs in a working set that is not actively being accessed, it could potentially have zero productivity cost to the system users: the proverbial tree fallen in a forest.

To leverage this working set locality, we introduce PERSES, a data allocation model designed to decrease the impact of device failures on the productivity and perceived availability of a storage system. PERSES is named for the titan of cleansing destruction in Greek mythology. We name our system PERSES because it focuses destruction in a storage system to a small number of users so that others may thrive. Our goal with PERSES is to isolate failures away from as many users as possible.

PERSES reduces the impact of failures by laying out data on a disk array according to dynamically computed working sets and selectively re-replicating data that is in multiple working sets. Since elements of a working set are co-located on disk, any single failure impacts a very small number of working sets. Working sets can either be derived from the metadata in the trace or calculated statistically based on the trace access pattern. Previous work has shown that algorithms such as N-Neighborhood Partitioning (NNP) can calculate statistical traces in O(n) [26, 27]. We use NNP to calculate working sets quickly without the overhead of maintaining groups by hand.

Multi-user cloud storage systems, the primary target of PERSES, are caught up in the data deluge, as many media sites and datasets explode in size. According to the 2011 IDC report, digital content is expected to grow to 2.7 zettabytes by the end of 2012, which would be a 48% increase over 2011 [5]. Cloud applications such as music players and document sharing rely on their user base experiencing a very low response time. In this environment, the cloud provider has a higher perceived availability if, on failure, their system appears slow or unavailable to a small number of users versus somewhat degraded to a large number. Other use cases that are disproportion-

ately affected by partial data degradation include compilation and some scientific workloads. 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 [20]. These systems resemble archives in that while there are distinct, consistent working sets, 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.

To demonstrate the benefits of PERSES, we arranged data onto simulated disks according to both pre-labeled and statistically derived groups and used our simulator to inject faults during a real data trace. We show that the PERSES layout leads to faults that affect fewer working sets 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 *group time lost*, which we define as the total delay encountered by all of data groups as a result of the access requests that are delayed while a disk is rebuilding.

We found that with a grouping that biased towards larger groups, PERSES gained over 4000 hours of group 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 while rebuild speed was a major factor in relative group time lost, adding a read cache made no noticeable difference. Finally, we discovered that setting a minimum group size improved the performance under PERSES even on smaller disks, with a maximum value of 94% less time lost.

Our main contributions are:

1. Methodology for working set selection for better performance during failure events (PERSES)
2. Fault simulator with real trace data showing up to 80% decrease in group time lost with PERSES allocation
3. Parameter identification and optimization for rebuild speed and minimum group size

After reviewing other work in the field, in Section 3 we describe our simulation design and sample workloads. We then, in Section 4 describe the results of our simulation work followed by an analysis of our results and the effect of scrubbing on our failure model. Finally, in Section 5 we discuss the implications of PERSES for large scale system design.

2 Background and Related Work

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 [15]. Other studies have shown that latent sector errors can lead to drive failure [4]. Pinheiro *et al.* showed that failures are both difficult to predict and common [12]. 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 [17].

Managing availability in RAID-like systems has received attention commensurate with increasing disk size and corresponding rebuild difficulty [8]. Disk sizes have grown, but the read speed is limited by power concerns and by the areal density of data on the platters [28]. Additionally, online-reconstruction is increasingly becoming CPU-bound, meaning that the cost of on-line reconstruction is unlikely to go down any time soon [10].

Modern storage systems can use parity to serve requests for data on a failed disk while the disk is rebuilding [6]. Many reconstruction optimizations have been proposed for distributed RAID rebuild [30, 31]. Online 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 [29].

WorkOut addresses this degradation by employing a surrogate RAID array and using that to serve requests for "popular" data where popular is defined as accessed twice within the course of the rebuild [29]. Tiair *et al.* also base their reconstruction algorithm on popularity [23]. These approaches are limited to workloads that have many repeat accesses to the same data in a very short period of time. PERSES sidesteps this limitation by exploiting correlation along multiple dimensions in addition to recency and frequency. Many other groups have shown significant improvement in performance during rebuild by modifying the underlying parity structure or scheduling [7, 22, 23]. We intend in future work to combine PERSES 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.

Generalized data layout is a major area of storage and filesystems research. Typically, the focus is on bringing the most recently active data to the best tier of storage, such as PROFS which does this for LFS [24]. Amur *et al.* look at layout for multi-replica systems in the context of power management [2]. 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 files that are members of multiple groups [9]. Sun *et al.* show that for parallel systems, there is a strong argument for application-driven

data placement [21]. However, none of these projects focus on layout for reconstruction.

Grouping data on disk provides benefits such as being able to avoid track boundaries [13], isolate faults [18], and avoid power consumption from excessive disk activity [11, 25]. Many of the grouping techniques in previous work rely on a high rate of repeat accesses, making those techniques unsuitable for PERSES [11]. On exascale systems, these benefits are magnified because every element in a group may be on separate disks, necessitating many extra spin-ups that both waste power and decrease the lifetime of the system [12, 25].

PERSES is inspired in particular by D-GRAID, which demonstrated that placing blocks of a file adjacent to each other on a disk reduced the system impact on failure, since adjacent blocks tend to fail together [18]. 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 person or project time lost as a result of file unavailability during the rebuild process. PERSES translates this idea to cloud and other large storage arrays.

Arpaci-Dusseau *et al.* have made a variety of advances in semantically aware disk systems [3, 19], but their online inference mechanism had trouble with the asynchronous nature of modern operating systems. We used suggestions from their work to categorically group our traces. We also examine how statistical classification methods compared to classification using metadata. We did not want to limit ourselves to a particular workload type, so use N-Neighborhood Partitioning (NNP) to do much of our working set calculation [26]. NNP has been shown to find working sets in real time as well as find predictive working sets. We chose this over other working set algorithms because we wanted PERSES to have low administrative overhead while not depending on popularity for classification. This type of grouping has been shown effective for cache management in online deduplication as well as power [26].

We built PERSES so that for statistical grouping we need only collect the bare minimum of data, which allows our algorithms to work almost domain-blind.

3 Design

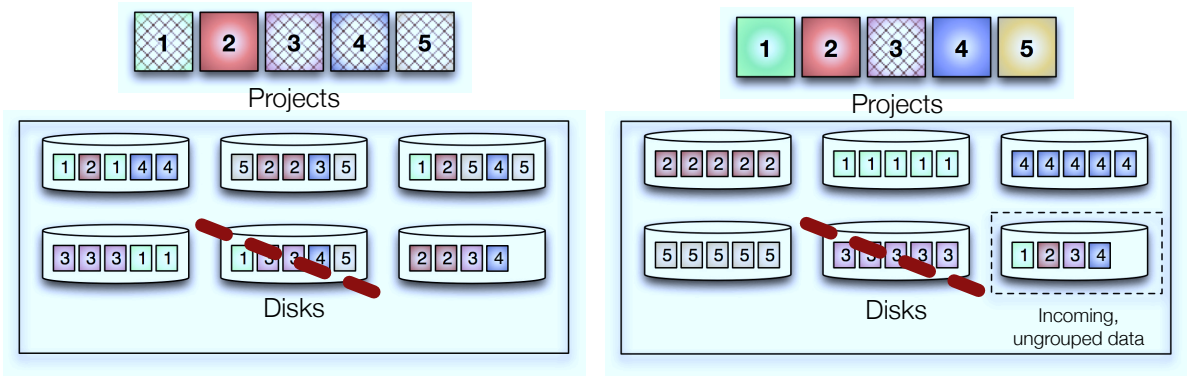
PERSES is designed as a generic technique for data allocation in a high-availability multi-purpose environment. Generally, when a disk fails, access to the data on the failed disk is degraded from having to reconstruct data from parity. PERSES is based on the idea that degraded access to a fraction of the data in a working set has a disproportionate impact to the working set as a whole. If a project on a failed disk is not accessed while the disk rebuilds, the failure is *unobserved*; it does not count towards the group time lost. On systems with highly bursty access patterns, unobserved failures become increasingly common as data is co-located. We also note that even if data is readable during rebuild, the performance can be degraded enough to slow down the entire project. In our simulation, we model a range of rebuild times to better understand the impact of PERSES in different reliability environments.

Consider a shared storage system that is holding data for five projects on six disks. Figure 1 shows what a failure would look like on this system with and without PERSES. The numbered boxes correspond to blocks of data, and the numbers correspond to projects. In the ungrouped example, Figure 1(a), data is arranged without any consideration to group 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 1(b), the data is laid out across the disks based on either access patterns or metadata. Groupings are refreshed periodically and the layout re-arranged, so some data will always be unlabeled. 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 unobserved.

PERSES is designed for a system with multiple disks, different projects that rely on many blocks of data, a controller to group data and manage layout, and 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 4.2 that this is not strictly necessary. We do not specify a particular reliability scheme, since even though PERSES is designed for parity-based reliability all reliability schemes save for mirroring introduce a reconstruction delay. We care about reconstruction speed but not method. Rebuild requests are typically given very low priority compared to incoming reads and writes, so a parity system should have little overhead that we do not account for [6]. We use the term *degradation* to refer to any delay in reading data that was on a failed disk, and we discuss failure from the perspective of working sets.

3.1 Working Sets

In order to lay out data by working set, we need to know what the working sets are. Manually maintaining working sets has high administrative overhead and presents a consistency problem [27]. PERSES determines working sets automatically, either by using metadata or by statis-



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

(b) With PERSES. Single failure impacts Project 3

Figure 1: 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.

tically analyzing access patterns. In either case, working sets are determined by analyzing a portion of the accesses and are then applied to the remainder. In a real-world setting, the working sets and parameters would be re-calculated at intervals determined by a running average of predictive power for the current grouping [26].

Previous work shows that in some long-term storage workloads, accesses happen in close temporal proximity within the same working sets [25]. They further show that this temporal proximity can be combined with spatial proximity to decrease the power footprint of large storage [25]. A data layout designed for fast reconstruction is a natural progression from a power aware data layout since both exploit highly correlated access patterns. Many workloads show an inverse Pareto distribution of group access probabilities over time [26]. This results in a high probability of an unobserved failure for small failure events, as any given working set has a low expected value for immediate future access. Working set access probabilities tend to have a shifting set of peak accessed working sets with a long tail of seldomly accessed working sets. With PERSES, failures in this long tail have a chance of having minimal to no system-wide impact.

Other projects examining grouping for storage have found that it is possible to group data in large systems such that the groups have predictive power, meaning that group membership implies a conditionally dependent probability of access within a given period of time [11, 26, 27]. There is also evidence that these groups can correspond to real-life working sets for applications, users, or projects [26]. We compare statistically derived groupings with categorical, metadata informed groupings to determine a fast and low-overhead algorithm for predictive data layout analysis.

Categorical groupings are groupings based on metadata or externally curated tags that describe data characteristics that an administrator believes to be important. We started with categorical grouping since prior work has suggested that using pre-defined groups is one of the best ways to investigate the impact of classifications [19].

Statistical groupings, on the other hand, are derived from extrapolating relationships based on a period of accesses and modifying these extrapolations as new data enters the system. This machine learning approach to grouping is a better fit for dynamic data where the meaning of curated labels drifts over time. Since many systems have strict privacy controls or other barriers to collecting metadata-rich traces, we investigated techniques to detect working sets with minimal private data. Wildani *et al.* showed that the NNP algorithm can form predictive groups using only LBA and timestamp of each access [26]. We adapted NNP to PERSES by doing a full parameter search since the parameters in Wildani *et al.* were optimized for small groups.

3.1.1 Statistical Grouping with NNP

Neighborhood Partitioning is a statistical method to compare data across multiple dimensions with a definable distance metric. Though it is very efficient and has some ability to detect interleaved groupings, it does not scale. To support arbitrarily large amounts of data, we use N-Neighborhood Partitioning (NNP), which merges several partitions without the memory overhead of a single large partitioning. By aggregating incoming accesses into regions of fixed size, NNP is highly scalable and able to perform in real time even in systems with high IOPS. The size of regions is determined by the memory capabilities of the system calculating the working sets, though increasing the size of the region quickly meets diminishing returns [27]. The regions in our implementation also overlap by a small number of accesses to account

for groups that straddle the arbitrary breakpoints in our region selection.

For each region, the partitioning steps are:

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

For n accesses in a region, 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, defined as

$$d(p_i, p_j) = d(p_j, p_i) = \sqrt{(t_i - t_j)^2 + s \times (o_i - o_j)^2}$$

where a point $p_i = (t_i, o_i)$, t = time, o = ID, and s is a scaling factor based on the typical relative distance between access IDs versus time.

We were most interested in recurring ID 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 block IDs in our data set. This matrix was calculated by identifying all the differences in timestamps

$$T = [T_1 = t_{i1} - t_{j1}, T_2 = t_{i1} - t_{j2}, T_3 = t_{i2} - t_{j1}, \ldots]$$

between the two IDs o_i and o_j. Weighting timestamps led to overfitting, so we decided to treat the unweighted average of these timestamp distances as the time element in our distance calculation. Thus, the distance between two IDs is:

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

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 group g_1, otherwise, it is placed into a new group g_2, and so on. Figure 2 illustrates a simple case.

3.1.2 Combining Neighborhood Partitions

A grouping G_i is a set of groups $g_1, \ldots, g_l$ that were calculated from the i^{th} region of accesses. Unlike plain 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 groups

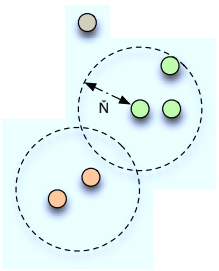


Figure 2: Each incoming access is compared to the preceding access to determine whether it falls within the neighborhood ($\check{N}$) to be in the same group. If it does not, a new group is formed with the incoming access.

within the groupings. So, for groupings $G_1, G_2, \ldots G_k$, the total grouping G is :

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

where the groupwise symmetric difference, Δ_g, 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 but not all. This group calculation happens in the background during periods of low activity. As accesses come in, we need to update groups to reflect a changing reality. We do this by storing a likelihood value for every group. This numerical value starts as the median intergroup distance value and is incremented when the grouping successfully predicts an access.

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.

3.1.3 Runtime

Neighborhood partitioning runs in $O(n)$ since it only needs to pass through each neighborhood twice: once to calculate the neighborhood threshold and again to collect the working sets. This makes it an attractive grouping mechanism for workloads with high IOPS, where a full $O(n^2)$ comparison is prohibitive. Additionally, we can capture groups in real time and quickly take advantage of correlations.

4 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 groupings
3. Lay out data across disks
4. Run through trace

Disk are initialized with all of the data that is read through the course of the trace. Disks are filled progressively with either grouped or randomly allocated data such that the only empty space is on the final disk. The amount of data each trace accesses is fixed, so as we add more disks to the simulator the size of disks decreases since the data is spread over more disks. Since we do not know the size of the files accessed in our trace, we allocate 10 MB per file. This number is entirely arbitrary, but the simulation results that use it can easily be translated to any real world system with fixed size blocks by adjusting the data to disk ratio accordingly. The simulator has an LRU read cache to capture popular accesses. The default cache starts cold and is 10 GB. This cache is assumed to be in memory and thus not part of our failure model other than to reduce the number of repeat disk accesses.

Groups are calculated before the trace is run, as described in Section 3.1. For grouped experiments, groups are then laid out sequentially on disks starting with the smallest groups. Random experiments have files laid out randomly without attention to group membership. Modeling correlated failure and doing a bin packed layout where large groups are paired with smaller to minimize total groups per disk is in our future work.

When disks are initialized, they are given a low uniform probability of failure. After each access, the probability of failure is increased by .001% to represent wear on the device [12]. Only full-disk failures are considered since recent work has shown latent sector errors to be surprisingly rare on modern hardware [14].

We express the rebuild speed of a disk 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 ($\approx$34 MB/s) as our default value for reconstructing data based on the 50 MB/s read speed of an off-the-shelf 7200 RPM disk [10]. This is a low estimate since CPU saturation and not read speed is the typical bottleneck for disk rebuild [10], and Section 4.2 shows that as r increases PERSES performs even better. We also test on r values as low as 10 s/GB ($\approx$102 MB/s) to demonstrate that PERSES can reduce group time lost even on faster hardware.

We define *group time lost* as the amount of time a group experiences degraded performance as a result of disk rebuilds. Group time lost can be higher than total system impact since a failure, especially on disks without working-set aware allocation, can affect multiple groups and each group contributes to group time lost. Once the data is laid out on disk, the trace is played back in the simulator to calculate the total system impact of all failure events and total group time lost. 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. All random layout runs were run at least 50 and typically over 100 times.

We do not include scrubbing in our simulation because scrubbing, which touches every element it can regardless of semantic qualities, does not satisfy our group association assumption [16]. Members of a group can be scrubbed even if a fraction of the group is offline. The only effect scrubbing is likely to have on group time lost is to slightly increase the failure rate caused by the total number of spin-ups of the drive.

4.1 Grouped Data Sets

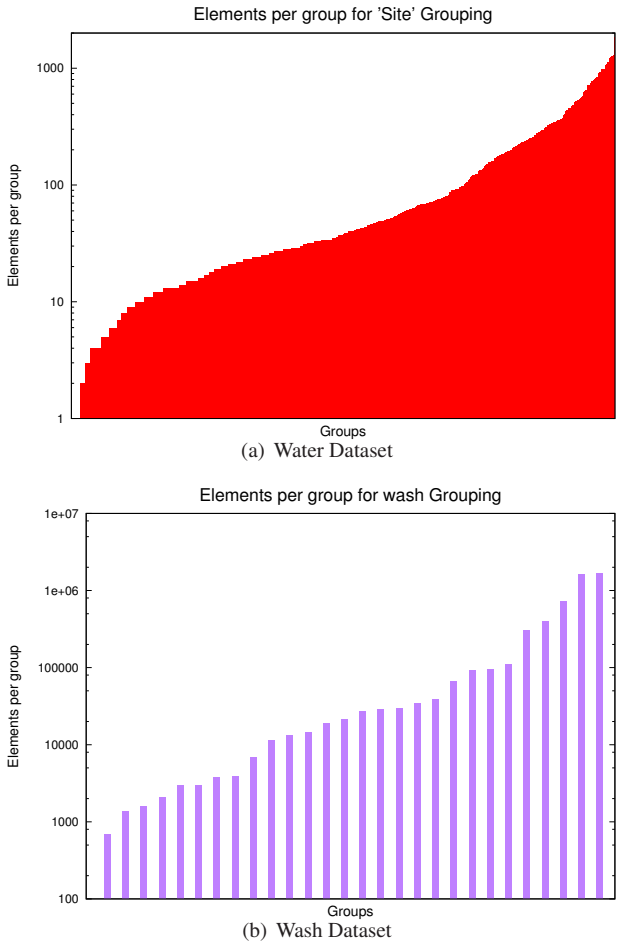
We use two traces in this work. We chose these traces because they had both accesses and semantic information, so we could for the first time provide a direct comparison between statistically and categorically defined groups.

The first data set is from the California Department of Water Resources. Our data consists of 90,000 accesses to a record store from 2007 through 2009. We make the assumption that queries correspond to record accesses. The data set is pre-grouped, and the grouping labels we consider for each access are "Timestamp," "Site," "Site Type," and "District." The dataset provided an additional grouping, "Year," that we chose not to use because it is inconsistently applied.

Our other data set is a database of vital records from the Washington state digital archives where records are labeled with one of many type identifiers (*e.g.* "Birth Records", "Marriage Records") [1]. We examined 5,321,692 accesses from 2007 through 2010. In addition to the supplied type identifiers, each record accessed had a static[1] RecordID that is assigned as Records are added to the system. We use these IDs as a second dimension when calculating statistical groupings, which are discussed in Section 3.1.1.

4.1.1 Categorical Grouping

We used principal components analysis to select the most predictive and automatically derivable feature for **water**, which was "Site." The single feature for **wash** was



(a) Water Dataset

(b) Wash Dataset

Figure 3: Grouping size distribution for Categorical Groups. Both datasets show a similar distribution of group sizes, with the **wash** dataset having much larger groups overall. Note that the y-axis is on a log scale and the x-axis corresponds to individual groups.

"Record Type." This is consistent with previous work on this data [1, 25]. Figure 3 shows the distribution of group sizes for categorically grouped traces of **water** and **wash** data. We see in Figure 3 that both graphs exhibit a logarithmic decrease in number of groups as group size increases.

4.1.2 Statistical Grouping

We used the NNP algorithm to identify statistical working sets in our **wash** trace. Groups are determined based on access times and a unique per record ID field that most record possessed. Those that were missing the ID field were treated as singletons. We could not statistically group the **water** trace because there was not enough information to provide a viable second continuous dimension for the classifier.

NNP parameters that affect the grouping include two scaling factors and weights for means and standard deviations in the partitioning calculation. Lacking any external confirmation of validity, we did a complete crawl through the parameter space and then calculated the average group size of all of the resultant groupings.

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

Table 1: Statistics of the two non-trivial groupings NNP found in **wash**

When these groupings were clustered using the parameters and average group size as features, elements fell into one of three clusters. The groupings within each cluster were almost identical. 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 1 shows the main differences between the two groupings. Though the difference in average group sizes seems small, the group size distribution within the grouping (Figure 4) shows that the inverse Pareto distribution of group sizes results in many more larger groups for the `wash-A` grouping.

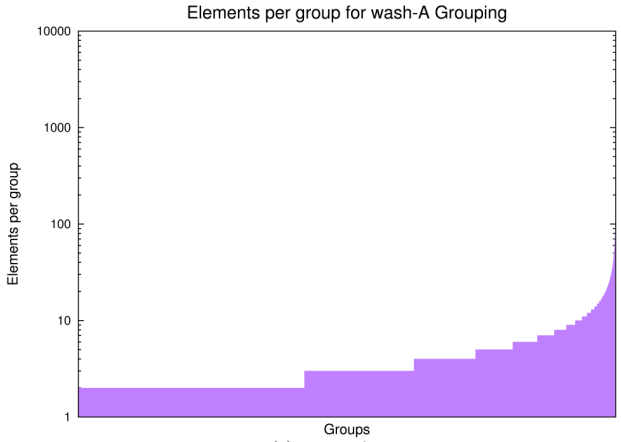
4.2 Simulation Results

We ran our PERSES simulator on categorically grouped **water** and **wash** data as well as the two statistical groupings of **wash** data. We saw the most improvement from the statistical groupings. To help understand the impact of PERSES, we also averaged 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*.

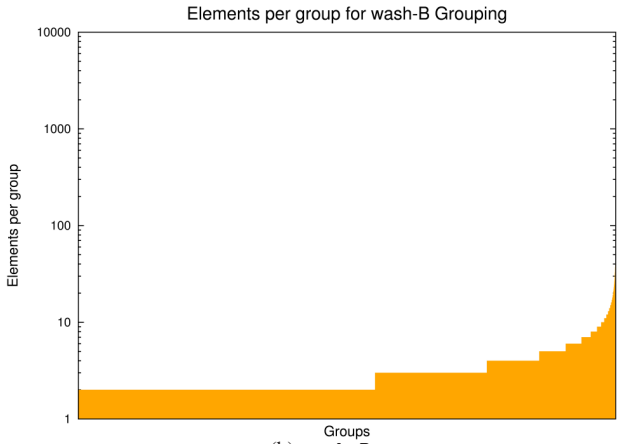
4.2.1 Categorical Groupings

We chose to use "Site" as main label for the **water** data after using principal components analysis to see which label was most predictive. Figure 5 shows that all of the labels provide very similar grouped time lost values over the course of the trace. This is surprising because grouped time lost is determined with respect to the grouping. So, over the course of the trace every grouping sees roughly equivalent impact for failures. This indicates that though the groups are predictive, there is not enough data to isolate groups.

Figure 6 shows that categorical grouping with "Site" on **water** does not significantly reduce the group time lost compared to random allocation for $r = 30$. Further-



(a) `wash-A`



(b) `wash-B`

Figure 4: Both `wash-A` and `wash-B` statistical groupings have enough very small groups to greatly skew the group size distribution towards larger groups. Groups sizes obey an inverse Pareto distribution.

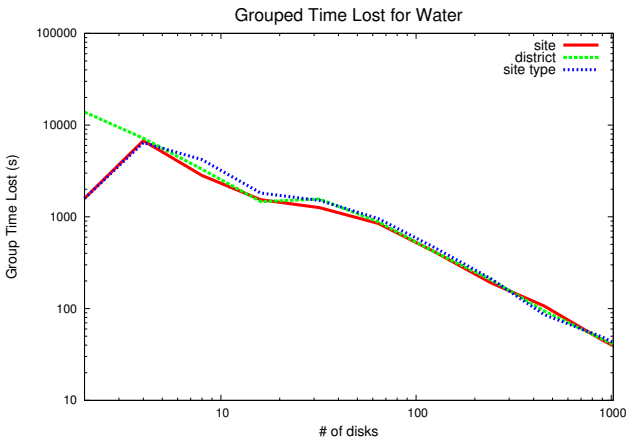
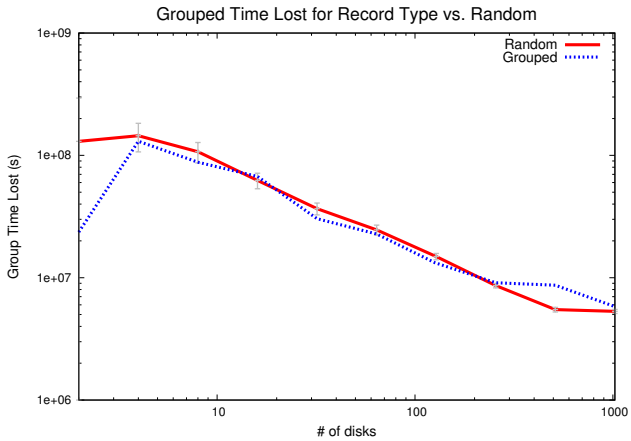
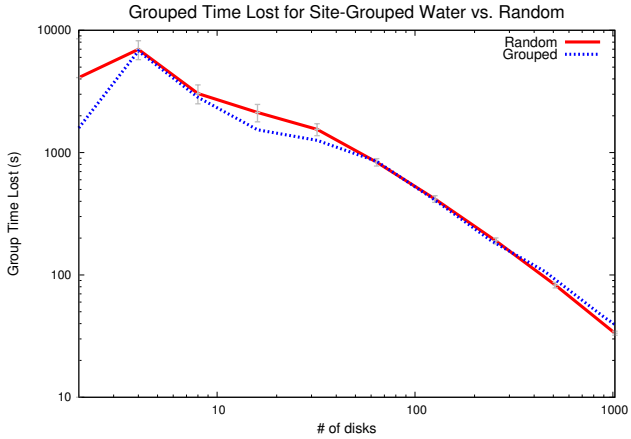


Figure 5: All of the categorical labels in the water trace provide similar grouped time lost trends.



(a) Washington: 1060 hours gained



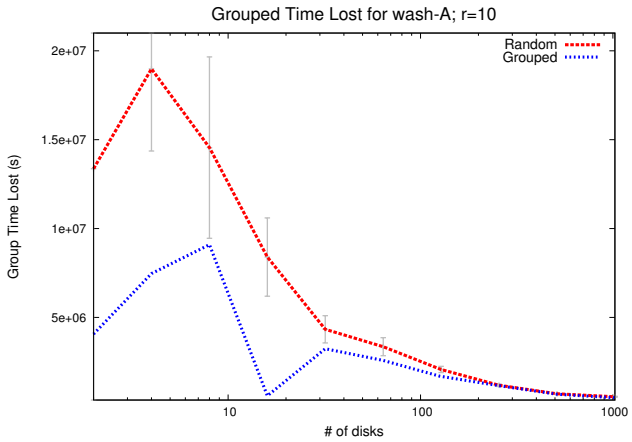
(b) Water: .09 hours gained

Figure 6: These graphs show that neither categorical grouping significantly improves on random arrangement at $r = 30$.

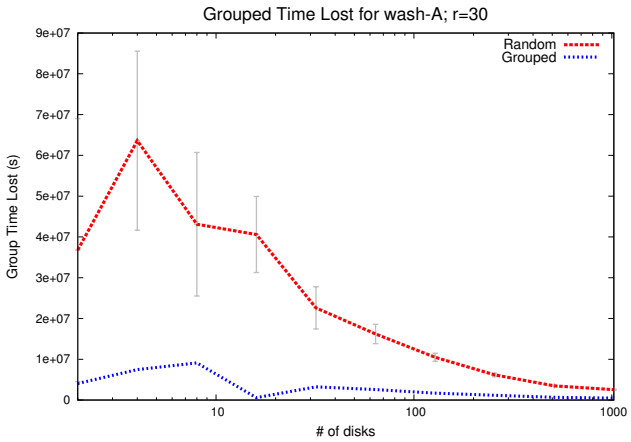
more, we found that increasing r, the number of seconds it takes to rebuild a gigabyte of data, had negligible effect on the relative group time lost. "Record Type" on **wash** performs better, saving about 1000 hours, largely due to the higher access rate for **wash**.

4.2.2 Statistical Groupings

Figure 7 shows how allocating data on disk using the `wash-A` grouping improves group 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 $r = 30$, which is the value we focus our tests on, 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 (Figure 8). This is because

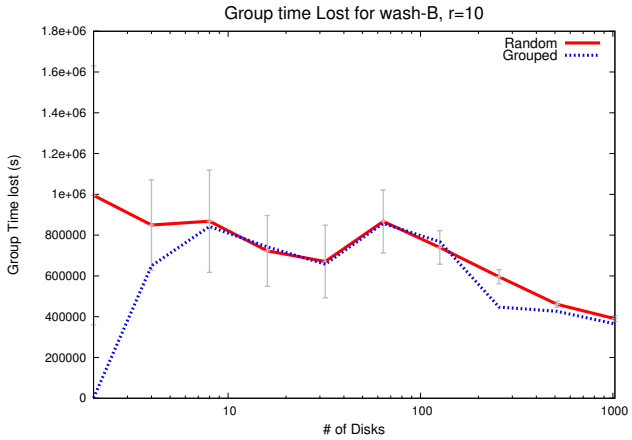


(a) r=10: 837 hours gained

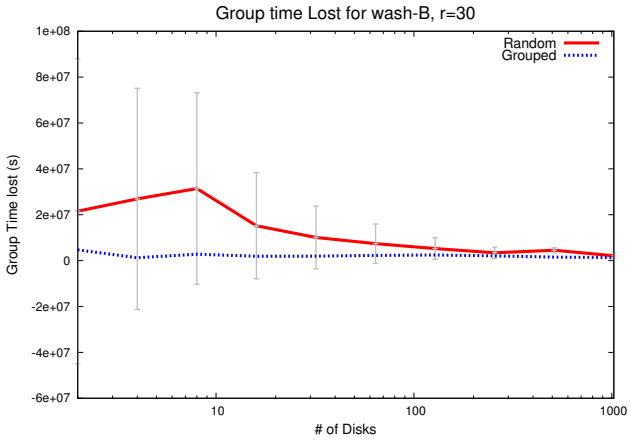


(b) r=30: 4289 hours gained

Figure 7: PERSES significantly reduces group time lost with the `wash-A` grouping.



(a) r=10: 12 hours gained



(b) r=30: 2740 hours gained

Figure 8: PERSES shows less improvement with the `wash-B` grouping.

`wash-B` has smaller, noisier groups, which biases the layout algorithm towards placing related data farther apart on average. We discuss this further in Section 5.

We compare against random since we have no information as to the actual layout of the data behind our traces. To give us another bound on PERSES performance, we also compared it against a temporal layout where , where files are placed on disk in the order they will later be read in the trace. Figure 9 shows that even against this optimal temporal layout, PERSES has less than or equal group time lost for all but the largest disks, with an average of 1300 hours saved over the three years of the trace.

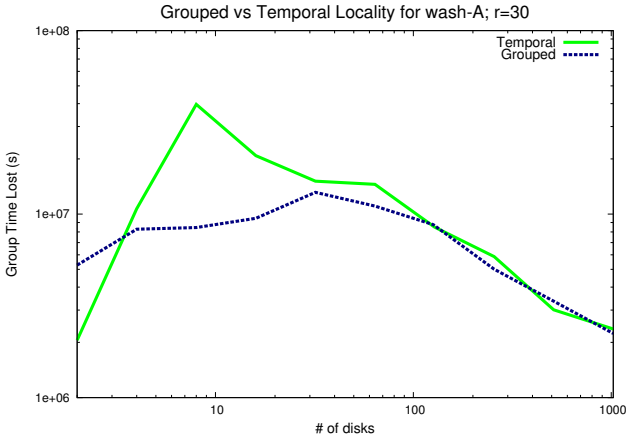


Figure 9: PERSES performs as well as or better then data allocated with a temporal oracle. 1321 hours gained.

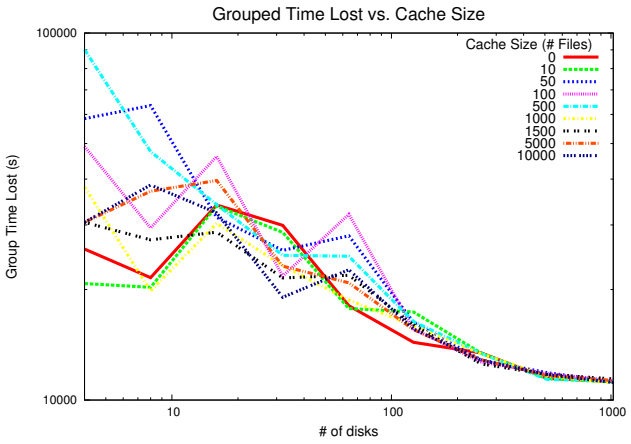


Figure 10: Cache size has negligible effect on group time lost. This graph was made with $r = 30$ and grouping `wash-B`, but other parameter combinations produced similar results.

4.2.3 Refinements

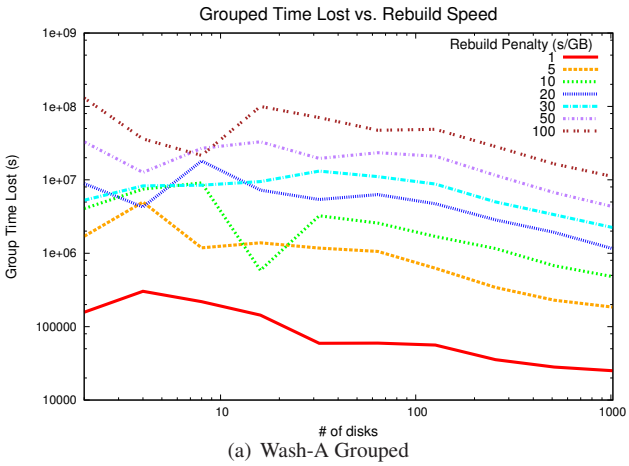
To further study our results, we explored the effects of different cache sizes, different r values, limiting group size, and increasing the IOPS in our trace. For cache and rebuild, we wanted to explore how PERSES would behave in different system configurations. The minimum group size experiments came from the observation that larger groups had more to lose and experienced more benefit from localization. We found that restricting group size did improve the performance of PERSES. Finally, we realize that the traces we have have relatively low IOPS, and we wanted to address concerns about the efficacy of PERSES in an active environment. To do this, we sped up our trace by a factor of ten and show that it PERSES improves group time lost even on the compressed trace.

Caching

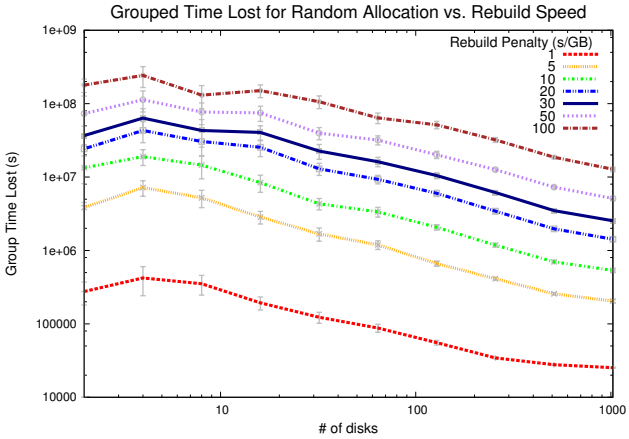
Our simulator has a read cache to catch popular reads before they go to disk. We found that the size of this cache, within reason, had very little impact on the group time lost. Figure 10 shows that not having any cache at all is competitive with having a cache of 10,000 files for **wash**, which represents 97 GB of cache. This trend continues until the cache becomes so large that every repeat request is served out of cache.

Rebuild Speed

Another question we had was how the rebuild factor affected group time lost. Figure 11 shows the improvement in group time lost for different rebuild factors. Though low r values do worse than random due to the overhead of disk arrangement, once values start approaching what disks can physically do, relative group time lost decreases as r rises.



(a) Wash-A Grouped



(b) Random

Figure 11: As the time to rebuild increases, PERSES grouping saves significant time over random

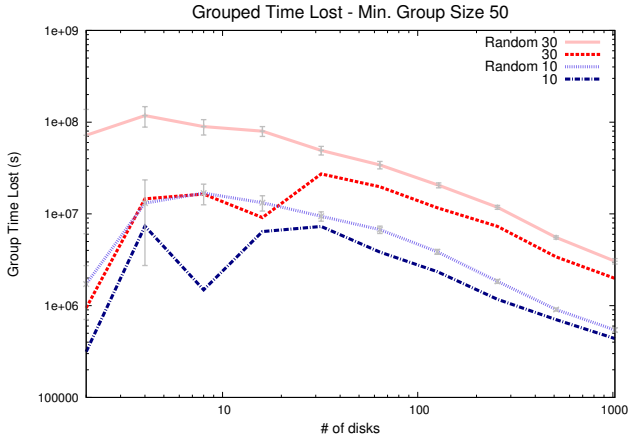
Minimum Group Size

After we noticed that `wash-A` outperformed `wash-B`, we ran experiments where we specified a minimum group size for our groupings. Our results indicate that PERSES has higher impact on larger groups since without group-based allocation there is a greater chance the project will be spread across many disks.

Figure 12 shows the `wash-A` grouping with minimum group sizes of 50 and 100 files. The top pair of lines on each graph are group allocated and random allocated data with $r = 30$ whereas the bottom pair are the same lines for $r = 10$. Surprisingly, if we restrict the size of groups to only model larger projects, we see significant improvement even with $r = 10$, which corresponds to a rebuild rate of 102 MB/s. This indicates that in a grouping with large groups, PERSES is valuable even on high-end hardware with very fast rebuild.

High IOPS

Finally, we had some concerns about how PERSES would behave under high IOPS. To test this, we compressed our



(a) Min. Group Size 50: 9252 hours gained for $r = 30$

(b) Min. Group Size 100: 9214 hours gained for $r = 30$

Figure 12: PERSES significantly lowers group time lost with a larger group size for `wash-A`. Group time lost is on a log scale

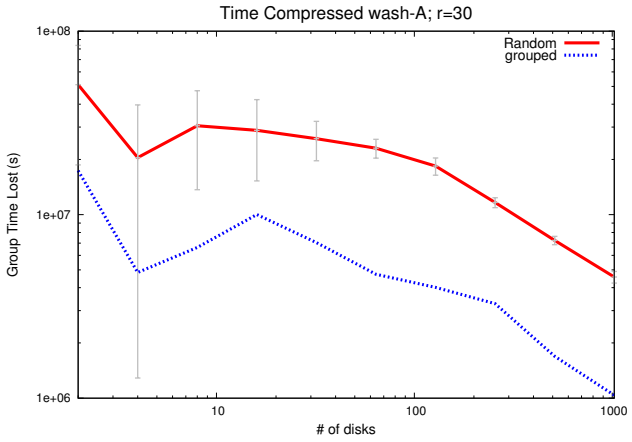


Figure 13: Compressing the accesses in `wash-A` by a factor of 10 only slightly hurts the performance of PERSES

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 13 shows that while PERSES does not save as much group time lost under compression, it still handily outperforms random allocation once disks are reasonably sized.

5 Discussion

Our results show a consistent reduction in group time lost for PERSES, and the reduction is greatest when groups are statistically derived and larger.

The most surprising result we found was that statistical grouping significantly outperformed categorical in terms of reducing group time lost. Statistical groupings outperform categorical groupings because, given the relatively short rebuild period, PERSES favors "tighter" groupings. By tighter, we mean groups where every group member has a high probability of co-access with every other group member. Statistical groups produced by NNP have this property, whereas the categorical groups in **wash** and **water** do not have any implicit promises of co-access. On the other hand, `wash-A` strictly outperforms `wash-B` because `wash-B` has groupings that are too tight. As a result, groups end up being very small on average and do not fully capture the co-access relationships in the trace data. We see this in Table 1 where `wash-A` has both larger groups and higher variance in group size than `wash-B`. The lesson is to bias NNP towards larger groupings for best PERSES performance.

In keeping with these two observations, it is unsurprising that we saw similar large gains when limiting the minimum group size of `wash-A` to 50 or to 100. PERSES gains the most benefit from placement of large groups while smaller groups are still more likely to be on a failed disk. We are working on distributing smaller groups across all disks to lower their probability of failure. Also, as the length of the rebuild increases, PERSES does much better because the longer a rebuild takes the more accesses are slowed, affecting more groups if the disk is not laid out according to group membership. Rebuild rate has a very high impact on the results because the rate between accesses is fixed by the timestamps in the trace. Thus, the probability that the failure will be observed by an access increases increases with downtime.

Surprisingly, adding a read cache did not significantly alter our results. This is because while elements were accessed in groups in our traces, they were almost never accessed repeatedly in short succession. This indicates that layout models based on the popularity of data for placement would have performed poorly on our traces. This sort of workload is prevalent in both record indices such as **wash** and scientific data such as **water**. These two datasets are by no means ideal; PERSES should perform

even better on data with higher IOPS and more distinct working sets such as virtual machine images or document servers.

One concern with grouping all of the data 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. We can mitigate this by arranging data on the tertiary replica such that in a multiple failure event, the tertiary datastore has a different arrangement and the data is preserved while retaining the benefit of increased availability in normal rebuild scenarios.

Rapid rebuild is touted as making RAID reconstruction irrelevant, but it has two major problems. First, the latent sector errors it fixes only happen in rapid succession, meaning that it will always be playing catch-up [15]. 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.

6 Conclusions

We have demonstrated PERSES, a data layout technique for significantly lower impact rebuilds in large multi-use storage systems. We have shown that we can automatically derive groupings from statistical data and apply them to lay out data such that a failed disk affects few working sets. Our experiments show that, with large automatically generated groupings, we can reduce the total time that groups perceive as lost by up to 80%, which corresponds to over 4000 hours over three years. We also showed that even against an optimal temporal layout, PERSES saves over 1300 hours in our three year trace.

Furthermore, we showed that PERSES is especially effective on larger groups, since even as the number of disks increase group time lost is still 40% lower with PERSES than without, and the average time gained is over 9000 hours. Finally, we showed that PERSES can operate with high IOPS, making it relevant for active systems including cloud storage. PERSES is general purpose and requires no administrative overhead to find groups or rearrange data, and it can combine with existing rapid reconstruction methods to help alleviate the growing constraint of disk rebuild.

6.1 Future Work

Our next steps are to add correlated failures to our fault simulator to better represent real 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 groups are allocated to disks. Currently, disks are filled in with groups based on group size. We are exploring using more intelligent bin packing to place groups 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 working sets and lay them out such that projects are isolated, we can control fragmentation in non-hierarchical systems without administrative overhead or time consuming metadata analysis.

7 Acknowledgments

This research was supported in part by the National Science Foundation, the Department of Energy, and industrial sponsors.

References

[1] ADAMS, I., STORER, M., AND MILLER, E. Analysis of workload behavior in scientific and historical long-term data repositories. *ACM Transactions on Storage (TOS) 8*, 2 (2012), 6.

[2] AMUR, H., CIPAR, J., GUPTA, V., GANGER, G., KOZUCH, M., AND SCHWAN, K. Robust and flexible power-proportional storage. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 217–228.

[3] ARPACI-DUSSEAU, A., ARPACI-DUSSEAU, R., BAIRAVASUNDARAM, L., DENEHY, T., POPOVICI, F., PRABHAKARAN, V., AND SIVATHANU, M. Semantically-smart disk systems: past, present, and future. *ACM SIGMETRICS Performance Evaluation Review 33*, 4 (2006), 29–35.

[4] BAIRAVASUNDARAM, L. N., GOODSON, G. R., PASUPATHY, S., AND SCHINDLER, J. An analysis of latent sector errors in disk drives. In *Proceedings of the 2007 SIGMETRICS Conference on Measurement and Modeling of Computer Systems* (June 2007).

[5] GENS, F. Idc predictions 2012: Competing for 2020. *IDC, Report, Dec* (2011).

[6] HOU, R., MENON, J., AND PATT, Y. Balancing i/o response time and disk rebuild time in a raid5 disk array. In *System Sciences, 1993, Proceeding of the*

Twenty-Sixth Hawaii International Conference on (1993), vol. 1, IEEE, pp. 70–79.

[7] JONES, L., REID, M., UNANGST, M., AND WELCH, B. Panasas tiered parity architecture. *Panasas White Paper* (2008).

[8] KRYDER, M., AND KIM, C. After hard driveswhat comes next? *Magnetics, IEEE Transactions on 45*, 10 (2009), 3406–3413.

[9] LAMEHAMEDI, H., SHENTU, Z., SZYMANSKI, B., AND DEELMAN, E. Simulation of dynamic data replication strategies in data grids. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International* (2003), IEEE, pp. 10–pp.

[10] LOUWRENTIUS. Raid array size and rebuild speed.

[11] PINHEIRO, E., AND BIANCHINI, R. Energy conservation techniques for disk array-based servers. In *ICS '04* (2004), ACM, pp. 68–78.

[12] PINHEIRO, E., WEBER, W., AND BARROSO, L. Failure trends in a large disk drive population. In *Proceedings of the 5th USENIX conference on File and Storage Technologies (FAST'07)* (2007).

[13] SCHINDLER, J., GRIFFIN, J., LUMB, C., AND GANGER, G. Track-aligned extents: matching access patterns to disk drive characteristics. In *Conference on File and Storage Technologies* (2002).

[14] SCHROEDER, B., DAMOURAS, S., AND GILL, P. Understanding latent sector errors and how to protect against them. *ACM Transactions on Storage (TOS) 6*, 3 (2010), 9.

[15] SCHROEDER, B., AND GIBSON, G. A. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *Proceedings of the 5th USENIX Conference on File and Storage Technologies (FAST)* (Feb. 2007), pp. 1–16.

[16] SCHWARZ, T. J. E., XIN, Q., MILLER, E. L., LONG, D. D. E., HOSPODOR, A., AND NG, S. Disk scrubbing in large archival storage systems. In *Proceedings of the 12th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '04)* (Oct. 2004), pp. 409–418.

[17] SCHWARZ, S. J., T., AND MILLER, E. L. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *Proceedings of the 26th International Conference on Distributed Computing Systems (ICDCS '06)* (Lisboa, Portugal, July 2006), IEEE.

[18] SIVATHANU, M., PRABHAKARAN, V., ARPACI-DUSSEAU, A., AND ARPACI-DUSSEAU, R. Improving storage system availability with D-GRAID. *ACM TOS 1*, 2 (2005), 133–170.

[19] SIVATHANU, M., PRABHAKARAN, V., POPOVICI, F., DENEHY, T., ARPACI-DUSSEAU, A., AND ARPACI-DUSSEAU, R. Semantically-smart disk systems. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies* (2003), pp. 73–88.

[20] STAELIN, C., AND GARCIA-MOLINA, H. Clustering active disk data to improve disk performance. *Princeton, NJ, USA, Tech. Rep. CS–TR–298–90* (1990).

[21] SUN, X., CHEN, Y., AND YIN, Y. Data layout optimization for petascale file systems. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage* (2009), ACM, pp. 11–15.

[22] THOMASIAN, A., TANG, Y., AND HU, Y. Hierarchical raid: Design, performance, reliability, and recovery. *Journal of Parallel and Distributed Computing* (2012).

[23] TIAIR, L., JIANG, H., FENG, D., XIN, H., AND SHIR, X. Implementation and evaluation of a popularity-based reconstruction optimization algorithm in availability-oriented disk arrays. In *Mass Storage Systems and Technologies, 2007. MSST 2007. 24th IEEE Conference on* (2007), IEEE, pp. 233–238.

[24] WANG, J., AND HU, Y. PROFS-performance-oriented data reorganization for log-structured file system on multi-zone disks. In *mascots* (2001), Published by the IEEE Computer Society, p. 0285.

[25] WILDANI, A., AND MILLER, E. Semantic data placement for power management in archival storage. In *Petascale Data Storage Workshop (PDSW), 2010 5th* (2010), IEEE, pp. 1–5.

[26] WILDANI, A., MILLER, E., AND RODEH, O. Hands: A heuristically arranged non-backup in-line deduplication system. Tech. Rep. UCSC-SSRC-12-03, University of California, Santa Cruz, Mar. 2012.

[27] WILDANI, A., MILLER, E., AND WARD, L. Efficiently identifying working sets in block i/o streams. In *Proceedings of the 4th Annual International Conference on Systems and Storage* (2011), p. 5.

[28] WOOD, R. Future hard disk drive systems. *Journal of magnetism and magnetic materials 321*, 6 (2009), 555–561.

[29] WU, S., JIANG, H., FENG, D., TIAN, L., AND MAO, B. Workout: I/o workload outsourcing for boosting raid reconstruction performance. In *Proceedings of the Seventh USENIX Conference on File and Storage Technologies (FAST09)* (2009).

[30] 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.

[31] XIN, Q., MILLER, E. L., AND SCHWARZ, T. J. E. 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, June 2004), pp. 172–181.

Notes

[1] This is not quite true, but accurate for our purposes. Further explanation can be found in [1].