Understanding latent sector errors and how to protect against them

Bianca Schroeder
Dept. of Computer Science
University of Toronto
Toronto, Canada
bianca@cs.toronto.edu

Sotirios Damouras
Dept. of Statistics
University of Toronto
Toronto, Canada
sotirios@utstat.toronto.edu

Phillipa Gill
Dept. of Computer Science
University of Toronto
Toronto, Canada
phillipa@cs.toronto.edu

Abstract

Latent sector errors (LSEs) refer to the situation where particular sectors on a drive become inaccessible. LSEs are a critical factor in data reliability, since a single LSE can lead to data loss when encountered during RAID reconstruction after a disk failure. LSEs happen at a significant rate in the field [1], and are expected to grow more frequent with new drive technologies and increasing drive capacities. While two approaches, data scrubbing and intra-disk redundancy, have been proposed to reduce data loss due to LSEs, none of these approaches has been evaluated on real field data.

This paper makes two contributions. We provide an extended statistical analysis of latent sector errors in the field, specifically from the view point of how to protect against LSEs. In addition to providing interesting insights into LSEs, we hope the results (including parameters for models we fit to the data) will help researchers and practitioners without access to data in driving their simulations or analysis of LSEs. Our second contribution is an evaluation of five different scrubbing policies and five different intra-disk redundancy schemes and their potential in protecting against LSEs. Our study includes schemes and policies that have been suggested before, but have never been evaluated on field data, as well as new policies that we propose based on our analysis of LSEs in the field.

1 Motivation

Over the past decades many techniques have been proposed to protect against data loss due to hard disk failures [3, 4, 8, 9, 14, 15, 18]. While early work focused on total disk failures, new drive technologies and increasing capacities have led to new failure modes. A particular concern are *latent sector errors (LSEs)*, where individual sectors on a drive become unavailable. LSEs are caused, for example, by write errors (such as a high-fly write) or by media imperfections, like scratches or smeared soft particles.

There are several reasons for the recent shift of attention to LSEs as a critical factor in data reliability. First and most importantly, a single LSE can cause data loss when encountered during RAID reconstruction after a disk failure. Secondly, with multi-terabyte drives using perpendicular recording hitting the markets, the frequency of LSEs is expected to increase, due to higher areal densities, narrower track widths, lower flying heads, and susceptibility to scratching by softer particle contaminants [6]. Finally, LSEs are a particularly insidious failure mode, since these errors are not detected until the affected sector is accessed.

The mechanism most commonly used in practice to protect against LSEs is a background scrubber [2, 12, 13, 17] that continually scans the disk during idle periods in order to proactively detect LSEs and then correct them using RAID redundancy. Several commercial storage systems employ a background scrubber, including, for example, NetApp's systems.

Another mechanism for protection against LSEs is intra-disk redundancy, i.e. an additional level of redundancy inside each disk, in addition to the inter-disk redundancy provided by RAID. Dholakia et al. [5, 10] recently suggested that intra-disk redundancy can make a system as reliable as a system without LSEs.

Devising effective new protection mechanisms or obtaining a realistic understanding of the effectiveness of existing mechanisms requires a detailed understanding of the properties of LSEs. To this point, there exists only one large-scale field study of LSEs [1], and no field data that is publicly available. As a result, existing work typically relies on hypothetical assumptions, such as LSEs that follow a Poisson process [2, 7, 10, 17]. None of the approaches described above for protecting against LSEs has been evaluated on field data.

This paper provides two main contributions. The first contribution is an extended statistical study of the data

in [1]. While [1] provides a general analysis of the data, we focus in our study on a specific set of questions that are relevant from the point of view of how to protect against data loss due to LSEs. We hope that this analysis will help practitioners in the field, who operate large-scale storage systems and need to understand LSEs, as well as researchers who want to simulate or analyze systems with LSEs and don't have access to field data. It will also give us some initial intuition on the real-world potential of different protection schemes that have been proposed and what other schemes might work well.

The second contribution is an evaluation of different approaches for protecting against LSEs, using the field data from [1]. Our study includes several intra-disk redundancy schemes (simple parity check schemes, interleaved parity [5, 10], maximum distance separable erasure codes, and two new policies that we propose) and several scrubbing policies, including standard sequential scrubbing, the recently proposed staggered scrubbing [13] and some new policies.

The paper is organized as follows. We provide some background information on LSEs and the data we are using in Section 2. Section 3 presents a statistical analysis of the data. Section 4 evaluates the effectiveness of intra-disk redundancy for protecting against LSEs and Section 5 evaluates the effectiveness of proactive error detection through scrubbing. We discuss the implications of our results in Section 6.

2 Background and Data

For our study, we obtained a subset of the data that was used by Bairavasundaram et al. [1]. While we refer the reader to [1] for a full description of the data, the systems they come from and the error handling mechanisms in those systems, we provide a brief summary below.

Bairavasundaram et al. collected data on disk errors on NetApp production storage systems installed at customer sites over a period of 32 months. These systems implement a proprietary software stack consisting of the WAFL filesystem, a RAID layer and the storage layer. The handling of latent sector errors in these systems depends on the type of disk request that encounters an erroneous sector and the type of disk. For enterprise class disks, the storage layer re-maps the disk request to another (spare) sector. For read operations, the RAID layer needs to reconstruct the data before the storage layer can remap it. For nearline disks, the process for reads is similar, however the remapping of failed writes is performed internally by the disk and transparent to the storage layer. All systems periodically scrub their disks to proactively detect LSEs. The scrub is performed using the SCSI verify command, which validates a sector's integrity without transferring data to the storage layer. A typical scrub interval is 2 weeks. Bairavasundaram et al. found that the majority of the LSEs in their study (more than 60%) were detected by the scrubber, rather than an application access.

In total the collected data covers more than 1.5 million drives and contains information on three different types of disk errors: latent sector errors, not-ready-condition-errors and recovered errors. Bairavasundaram et al. find that a significant fraction of drives (3.45%) develops latent sector errors at some point in their life and that the fraction of drives affected by LSEs grows as disk capacity increases. They also study some of the temporal and spatial dependencies between errors and find evidence of correlations between the three different types of errors.

For our work, we have been able to obtain a subset of the data used in [1]. This subset is limited to information on latent sector errors (no information on not-ready-condition-errors and recovered errors) and contains for each drive that developed LSEs information on the time when the error was detected and the logical block number of the sector that was affected. Note that since LSEs are by definition *latent* errors, i.e. errors that are unknown to the system until it tries to access the affected sector, we cannot know for sure when exactly the error happened. The timestamps in our data refer to the time when the error was detected, not necessarily when it first happened. We can, however, narrow down the time of occurrence to a 2-week time window: since the scrub interval in NetApp's systems is two weeks, any error must have happened within less than two weeks before the detection time. For applications in this paper where the timestamp of an error matters we use three different methods for approximating timestamps, based on the above observation, in addition to using the timestamps directly from the trace. We describe the details in Section 5.1.

We focus in our study on drives that have been in the field for at least 12 months and have experienced at least one LSE. We concentrate on the four most common nearline drive models (the models referred to as A-1, D-2, E-1, E-2 in [1]) and the four most common enterprise drive models (k-2, k-3, n-3, and o-3). In total, the data covers 29,615 nearline drives and 17,513 enterprise drives.

3 Statistical properties of LSEs

We begin with a study of several statistical properties of LSEs. Many baseline statistics, such as the frequency of LSEs and basic temporal and spatial properties, have been covered by Bairavasundaram et al. in [1], and we are not repeating them here. Instead we focus on a specific set of questions that is relevant from the point of view of how to protect against data loss due to LSEs.

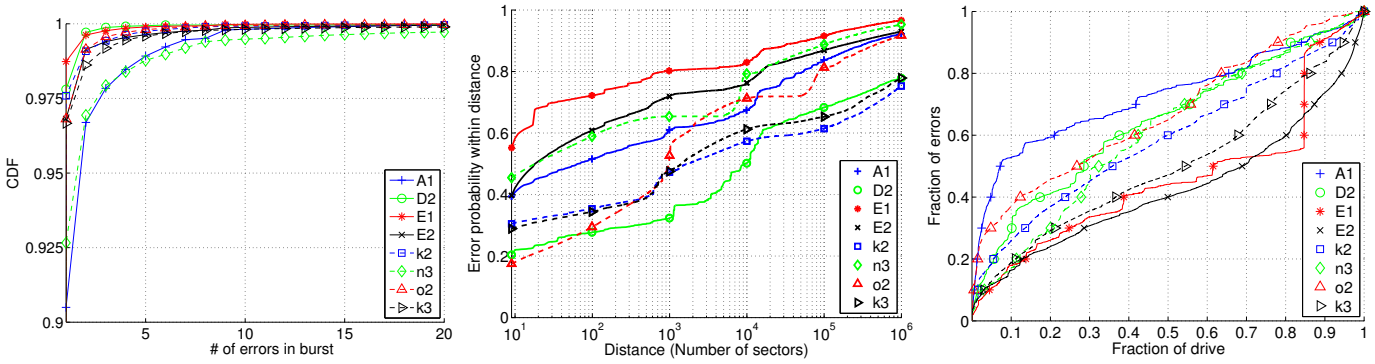

Figure 1: *Distribution of the number of contiguous errors in a burst (left), cumulative distribution function of the sector distance between errors that occur within same 2-week interval (middle), and the location of errors on the drive (right)*

3.1 How long are error bursts?

When trying to protect against LSEs, it is important to understand the distribution of the lengths of error bursts. By an error burst we mean a series of errors that is contiguous in logical block space. The effectiveness of intra-disk redundancy schemes, for example, depends on the length of bursts, as a large number of contiguous errors likely affects multiple sectors in the same parity group preventing recovery through intra-disk redundancy.

Figure 1(left) shows for each model the cumulative distribution function of the length of error bursts. We observe that in 90–98% of cases a burst consists of one single error. For all models, except A-1 and n-3, less than 2.5% of runs consist of two errors and less than 2.5% have more than 2 errors.

An interesting question is how to best model the length of an error burst and the number of good sectors that separate two bursts. The most commonly used model is a geometric distribution, as it is convenient to use and easy to analyze. We experimented with 5 different distributions (Geometric, Weibull, Rayleigh, Pareto, and Lognormal), that are commonly used in the context of system reliability, and evaluated their fit through the total squared differences between the actual and hypothesized frequencies (χ^2 statistic). We found consistently across all models that the geometric distribution is a poor fit, while the Pareto distribution provides the best fit. For the length of the error bursts, the deviation of the geometric from the empirical distribution was more than 13 times higher than that of the Pareto (13.50 for nearline and 14.34 for enterprise), as measured by the χ^2 statistic. For the distance between bursts the geometric fit was even worse. The deviation under the geometric distribution compared to the Pareto distribution is 46 and 110 times higher for nearline and enterprise disks, respectively. The geometric distribution proved such a poor fit because it failed to capture the long tail behavior of the data, i.e. the presence of long error bursts and the clustering of errors.

The top two rows in Table 1 summarize the parameters for the Pareto distribution that provided the best fit. For the number of good sectors between error bursts the parameter in the table is the α parameter of the Pareto distribution. For modeling the burst lengths we used two parameters. The first parameter p gives the probability that the burst consists of a single error, i.e. $(1-p)$ is the probability that an error burst will be longer than one error. The second parameter is the α parameter of the Pareto distribution that best fits the number of errors in bursts of length > 1.

3.2 How far are errors spaced apart?

Knowing at what distances errors are typically spaced apart is relevant for both scrubbing and intra-disk redundancy. For example, errors that are close together in space are likely to affect several sectors in the same parity group of an intra-disk redundancy scheme. If they also happen close together in time it is unlikely that the system has recovered the first error before the second error happened.

Figure 1 (middle) shows the cumulative distribution function (CDF) of the distance between an error and the closest neighbor that was detected within a 2-week period (provided that there was another error within 2 weeks from the first). We chose a period of 2 weeks, since this is the typical scrub interval in NetApp's filers.

Not surprisingly we find that very small distances are the most common. Between 20–60% of all errors have a neighbor within a distance of less than 10 sectors in logical sector space. However, we also observe that almost all models have pronounced "bumps" (parts where the CDF is steeper) indicating higher probability mass in these areas. For example, model o-2 has bumps at distances of around 10^3 and 10^5 sectors. Interestingly, we also observe that the regions where bumps occur tend

Variable	Dist./Params.	A-1	D-2	E-1	E-2	k-2	k-3	n-3	o-2
Error burst length	Pareto p, α	0.9, 1.21	0.98, 1.79	0.98, 1.35	0.96, 1.17	0.97, 1.2	0.97, 1.15	0.93, 1.25	0.97, 1.44
Distance btw. bursts	Pareto α	0.008	0.022	0.158	0.128	0.017	0.00045	0.077	0.05
#LSEs in 2 weeks	Pareto α	0.73	0.93	0.63	0.82	0.80	0.70	0.45	0.22
#LSEs per drive	Pareto α	0.58	0.81	0.34	0.44	0.63	0.58	0.31	0.11

Table 1: *Parameters from distribution fitting*

to be consistent for different models of the same family. For example, the CDFs of models E-1 and E-2 follow a similar shape, as do the CDFs for models k-2 and k-3. We therefore speculate that some of these distances with higher probability are related to the disk geometry of a model, such as the number of sectors on a track.

3.3 Where on the drive are errors located?

The next question we ask is whether certain parts of the drive are more likely to develop errors than others. Understanding the answer to this question might help in devising smarter scrubbing or redundancy schemes that employ stronger protection mechanisms (e.g. more frequent scrubbing or stronger erasure codes) for those parts of the drive that are more likely to develop errors.

Figure 1 (right) shows the CDF of the logical sector numbers with errors. Note that the X-axis does not contain absolute sector numbers, since this would reveal the capacity of the different models, information that is considered confidential. Instead, the X-axis shows percentage of the logical sector space, i.e. the point (x,y) in the graph means that y% of all errors happened in the first x% of the logical sector space.

We make two interesting observations: The first part of the drive shows a clearly higher concentration of errors than the remainder of the drive. Depending on the model, between 20% and 50% of all errors are located in the first 10% of the drive's logical sector space. Similarly, for some models the end of the drive has a higher concentration. For models E-2 and k-3, 30% and 20% of all errors, respectively, are concentrated in the highest 10% of the logical sector space. The second observation is that some models show three or four "bumps" in the distribution that are equidistant in logical sector space (e.g. model A-1 has bumps at fractions of around 0.1, 0.4 and 0.7 of the logical sector space).

We speculate the areas of the drive with an increased concentration of errors might be are areas with different usage patterns, e.g. filesystems often store metadata at the beginning of the drive.

3.4 What is the burstiness of errors in time?

While Bairavasundaram et al. [1] provide general evidence of temporal locality between errors, the specific question we are interested in here is how quickly exactly the probability of seeing another error drops off with time and how errors are distributed over time. Understanding the conditional probability of seeing an error in a month, given that there was an error x months ago, is useful for scrubbing policies that want to adapt the scrubbing rate as a function of the current probability of seeing an error.

To answer the question above, Figure 2 (left) considers for each drive the time of the first error and shows for each subsequent 2-week period the probability of seeing an additional error. We chose 2-week intervals, since this is the typical scrubbing interval in NetApp's systems, and hence the resolution of the error detection time. We observe that after the first month after the first error is detected, the probability of seeing additional errors drops off exponentially (note the log-scale on the Y-axis), dropping close to 1% after only 10 weeks and below 0.1% after 30 weeks.

Figure 2 (middle) illustrates how errors are distributed over time. We observe each drive for one year after its first error and count how many 2-week scrub intervals in this time period encounter any errors. We observe that for 55–85% of drives, all errors are concentrated in the same 2-week period. Only 10–15% of drives experience errors in two different 2-week periods, and for most models less than 15% see errors in more than two 2-week periods.

Summarizing the above observations, we find that the errors a drive experiences occur in a few short bursts, i.e. errors are highly concentrated in a few short time intervals. One might suspect that this bursty behavior is poorly modeled by a Poisson process, which is often used in modeling LSE arrivals [2, 7, 10, 17]. The reason for the common use of Poisson processes in modeling LSEs is that they are easy to analyze and that so far little data has been available that allows the creation of more realistic models. We fitted a Poisson distribution to the number of errors observed in a 2-week time interval and to the number of errors a drive experiences during its lifetime, and found the Poisson distribution to be a poor fit in both cases. We observe that the empirical distribution has a significantly longer tail than a Poisson distribution, and find that instead a Pareto distribution is a much better fit. For illustration, Figure 2 (right) shows for model n-3 the empirical distribution for the number of errors in a disks's lifetime and the Poisson and Pareto distributions

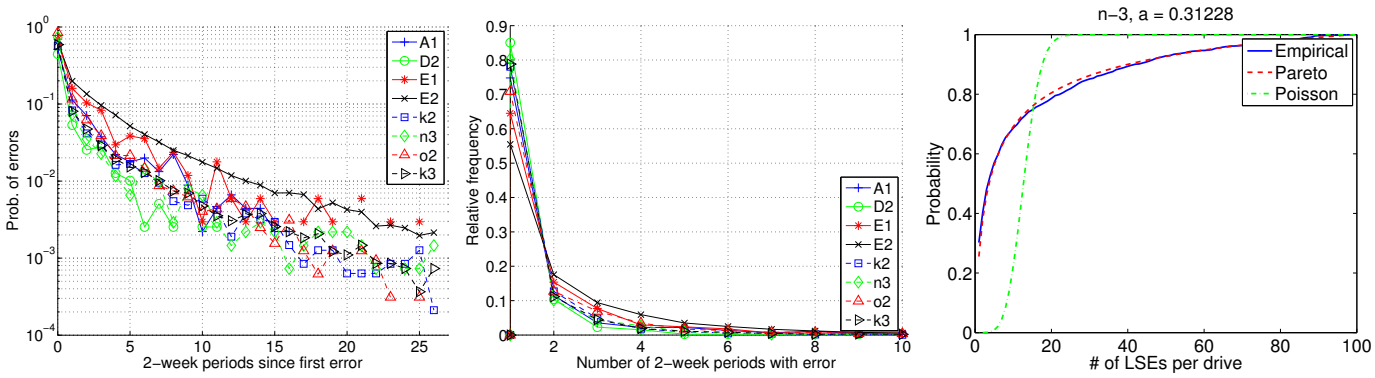

Figure 2: *The probability of seeing an error x 2-week periods after first error (left), the number of 2-week periods in a disk's life with at least one error (middle), and the distribution of the number of errors a disk sees in its lifetime (right).*

fitted to it. We provide the Pareto α parameter for both empirical distributions for all models in Table 1.

3.5 What causes LSEs?

This is obviously a broad question that we cannot hope to answer with the data we have. Nevertheless, we want to address this question briefly, since our observations in Section 3.3 might lead to hasty conclusions. In particular, a possible explanation for the concentration of errors in certain parts of the drive might be that these areas see a higher utilization. While we do not have access to workload data for NetApp's systems, we have been able to obtain two years of data on workload, environmental factors and LSE rates for five large ($> 50,000$ drives each) clusters at Google containing five different drive models. None of the clusters showed a correlation between either the number of reads or the number of writes that a drive sees (as reported by the drive's SMART parameters) and the number of LSEs it develops. We plan a detailed study of workload and environmental factors and how they impact LSEs as part of future work.

3.6 Does close in space mean close in time?

Prior work [1] and the questions above have focused on spatial and temporal correlations in isolation. For most error protection schemes, it is crucial to understand the relationship between temporal and spatial correlation. For example, for intra-disk redundancy schemes it does not only matter how long a burst of errors is (i.e. the number of consecutive errors in the burst), but also how much time there is between errors in a burst. More time between errors increases the chance that the first error is detected and corrected before the second error happens.

Figure 3 (left) shows the distribution of the time an error burst spans, i.e. the time difference between the first and last error in a burst. We observe that in more than 90% of the bursts the errors are discovered within the same 2-week scrub interval and in more than 95% of bursts the errors are detected within a month from each other. Less than 2% of error bursts span more than 3 months. These observations indicate that the errors in most bursts are likely caused by the same event and hence occurred at the same time.

Figure 3 (right) shows a more general view of the correlation between spatial and temporal locality. The graph shows for radii ranging from one sector to 50GB two bars: the first gives the probability that an error has at least one neighbor within this radius at some point during the disk's lifetime; the second bar gives the probability that an error has at least one neighbor within this radius within 2 weeks of time. As the graph shows, for small radii the two bars are virtually identical, indicating that errors that happened close in space were likely caused by the same event and hence happened at nearly the same time. We also observe that even for larger radii the two bars are still very close to each other. The figure shows results for model n-3, but we found results to be similar for all other models.

4 Protecting against LSEs with Intra-disk Redundancy

While inter-disk redundancy has a long history [3, 4, 8, 9, 14, 14, 15, 18], there are much fewer instances of intra-disk redundancy. Some filesystems [11] create in-disk replicas of selected metadata, IRON file systems [16] suggest to add a parity block per file, and recent work by Dholakia et al. [5, 10] introduces a new intra-disk redundancy scheme for all data blocks in a drive.

The motivation behind intra-disk redundancy is to reduce data loss when LSEs are encountered during RAID reconstruction, or where there is no inter-disk redundancy available. Dholakia et al. [5, 10] predict that with

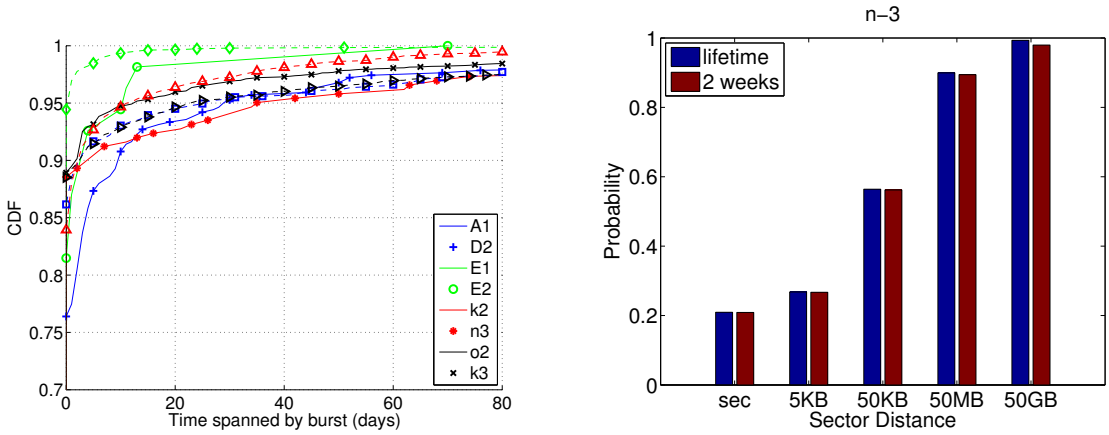

Figure 3: *Distribution of the time spanned by an error burst (left), and comparison of the probability of seeing another error within radius x in the 2 weeks after first error versus entire disk life (right)*

the use of intra-disk redundancy a system could achieve essentially the same reliability as that of a system operating without LSEs. Highly effective intra-disk redundancy might obviate the need for a background scrubber (and its potential impact on foreground traffic); in the best case, they might also enhance the reliability of a single parity RAID system sufficiently to make the use of double parity (e.g. RAID-4 or RAID-5) unnecessary, thereby avoiding the overheads and additional power usage of the second parity disk.

The intra-disk redundancy schemes we consider divide a disk into segments of k contiguous data sectors followed by m redundant sectors. The m redundant sectors are typically obtained using XOR-based operations on the data sectors. Different schemes vary in their reliability guarantees and their overhead depending on how the parity sectors are computed.

In our work, we evaluate 5 different intra-disk redundancy schemes. Three of the schemes (SPC, MDS, IPC) have been previously proposed, but have never been evaluated on field data. Two of the schemes are new schemes (MDS+SCP, CDP) that we suggest based on results from Section 3. All schemes are described below. We would like to note at this point, that while we do discuss the difference in overheads introduced by the different schemes, the focus of this section is to compare the relative degree of protection they can offer, rather than a detailed evaluation of their impact on performance.

Single parity check (SPC): A $k+1$ SPC scheme stores for each set of k contiguous data sectors one parity sector (typically a simple XOR on all data sectors). We refer to the set of k contiguous data sectors and the corresponding parity sector as a *parity group*. SPC schemes can tolerate a single error per parity group. Recovery from multiple errors in a parity group is only possible if there's an additional level of redundancy outside the disk (e.g. RAID). SPC schemes are simple and have little I/O overhead, since a write to a data sector requires only one additional write (to update the corresponding parity sector). However, a common concern is that due to spatial locality among sector errors, an error event will frequently affect multiple sectors in the same parity group.

Maximum distance separable (MDS) erasure codes: A $k+m$ MDS code consisting of k data sectors and m parity sectors can tolerate the loss of any m sectors in the segment. A well-known member of this code family are Reed-Solomon codes. While MDS codes are stronger than SPC they also create higher computational overheads (for example in the case of Reed-Solomon codes involving computations on Galois fields) and higher I/O overheads (for each write to a data sector all m parity sectors need to be updated). In most environments, these overheads make MDS codes impractical for use in intra-disk redundancy. Nevertheless, MDS codes provide an interesting upper bound on what reliability levels one can hope to achieve with intra-disk redundancy.

Interleaved parity check codes (IPC): A scheme proposed by Dholakia et al. [5, 10], specifically for use in intra-disk redundancy with lower overheads than MDS, but potentially weaker protection. The key idea is to ensure that the sectors within a parity group are spaced further apart than the length m of a typical burst of errors. A $k+m$ IPC achieves this by dividing k consecutive data sectors into $l = k/m$ segments of size m each, and imagining the $l \times m$ sectors $s_1, \ldots, s_{l \times m}$ layed out rowwise in an $l \times m$ matrix. Each one of the m parity sectors is computed as an XOR over one of the columns of this imaginary matrix, i.e. parity sector p_i is an XOR of $s_i, s_{i+m}, s_{i+2m}, \ldots, s_{i+(l-1)m}$. We refer to the data sectors in a column and the corresponding parity sector as a *parity*

Data Disk	Data Disk	Data Disk	Data Disk	Row Par. Disk	Diag. Par. Disk
0 (s_0)	1 (s_4)	2 (s_8)	3 (s_{12})	4 (p_0)	0 (p_4)
1 (s_1)	2 (s_5)	3 (s_9)	4 (s_{13})	0 (p_1)	1 (p_5)
2 (s_2)	3 (s_6)	4 (s_{10})	0 (s_{14})	1 (p_2)	2 (p_6)
3 (s_3)	4 (s_7)	0 (s_{11})	1 (s_{15})	2 (p_3)	3 (p_7)

Table 2: *Illustration of how to adapt RAID R-DP [4] with $p = 5$ for use in our intra-disk redundancy scheme CDP. The number in each block denotes the diagonal parity group a block belongs to. The parentheses show how an intra-disk redundancy segment with data sectors $s_0, ..., s_{15}$ and parity sectors $p_0, ..., p_7$ is mapped to the blocks in R-DP.*

group, and the $l \times m$ data sectors and the m parity sectors together as a *parity segment*. Observe, that all sectors in the same parity group have a distance of at least m. IPC can tolerate up to m errors provided they all affect different columns (and therefore different parity groups), but IPC can tolerate only a single error per column.

Hybrid SPC and MDS code (MDS+SPC): This scheme is motivated by Section 3.3, where we observed that for many models a disproportionately large fraction of all errors is concentrated in the first 5-15% of the logical block space. This scheme therefore uses a stronger (MDS) code for this first part of the drive, and a simple 8+1 SPC for the remainder of the drive.

Column Diagonal Parity (CDP): The motivation here is to provide a code that can tolerate a more diverse set of error patterns than IPC, but with less overhead than MDS. Our idea is to adapt the row-diagonal parity algorithm (R-DP) [4], which was developed to tolerate double disk failures in RAID, for use in intra-disk redundancy. R-DP uses $p+1$ disks, where p is a prime number, and assigns each data block to one row parity set and one diagonal parity set. R-DP uses $p-1$ disks for data, and two disks for row and diagonal parity. Figure 2 illustrates R-DP for $p = 5$. The row disk holds the parity for each row, and the number in each block denotes the diagonal parity group that the block belongs to.

We translate an R-DP scheme with parameter p to an intra-disk redundancy scheme with $k = (p-1)^2$ data sectors and $m = 2(p-1)$ parity sectors by mapping sectors to blocks as follows. We imagine traversing the matrix in Figure 2 column-wise and assigning the data sectors $s_0, ..., s_{15}$ consecutively to the blocks in the data disks and the parity sectors $p_0, ..., p_7$ to the blocks in the parity disks. The resulting assignment of sectors to blocks is shown in parentheses in the figure. Observe that without the diagonal parity, this scheme is identical to IPC: the row-parity of R-DP corresponds to to the parity sectors that IPC computes over the columns of the $(p-1) \times (p-1)$ matrix formed by rows of the data sectors. We therefore refer to our scheme as the column-diagonal parity (CDP) scheme.

CDP can tolerate any two error bursts of length $p-1$ that remove two full columns in Figure 2 (corresponding to two total disk failures in the R-DP scheme). In addition, CDP can tolerate a large number of other error patterns. Any data sector, whose corresponding column parity group has less than two errors or whose diagonal parity group has less than two errors, can be recovered[1]. Moreover, in many cases it will be possible to recover sectors where both the column parity group and the diagonal parity group have multiple errors, e.g. if the other errors in the column parity group can be recovered using their respective diagonal parity.

Note that for all codes there is a trade-off between the storage efficiency (i.e. $k/(k+m)$), the I/O overheads and the degree of protection a code can offer, depending on its parameter settings. Codes with higher storage efficiency generally have lower reliability guarantees. For a fixed storage efficiency, codes with larger parity segments provide stronger reliability for correlated errors that appear in bursts. At the same time, larger parity segments usually imply higher I/O overheads, since data sectors and the corresponding parity sectors are spaced further apart, requiring more disk head movement for updating parity sectors. The different schemes also differ in the flexibility that their parameters offer in controlling those trade-offs. For example, CDP cannot achieve any arbitrary combination of storage efficiency and parity segment size, since its only parameter p controls both the storage efficiency and the segment size.

4.1 Evaluation of redundancy schemes

4.1.1 Simple parity check (SPC) schemes

The question we want to answer in this section is what degree of protection simple parity check schemes can provide. Towards this end we simulate SPC schemes with varying storage efficiency, ranging from 1+1 to 128+1 schemes. While we explore the whole range of k from 1 to 128, in most applications the low storage efficiency of codes with values of k below 8 or 9 would probably render them impractical. Figure 4 shows the fraction of disks with uncorrectable errors (i.e. disks that have at least one parity group with multiple errors), the fraction of parity groups that have multiple errors, and the number of sectors per disk that cannot be recovered with SPC redundancy.

We observe that for values of k in the practically feasible range, a significant fraction of drives (about a quar-

[1]Exceptions are sectors in the diagonal parity group $p-1$, as R-DP stores no parity for this group.

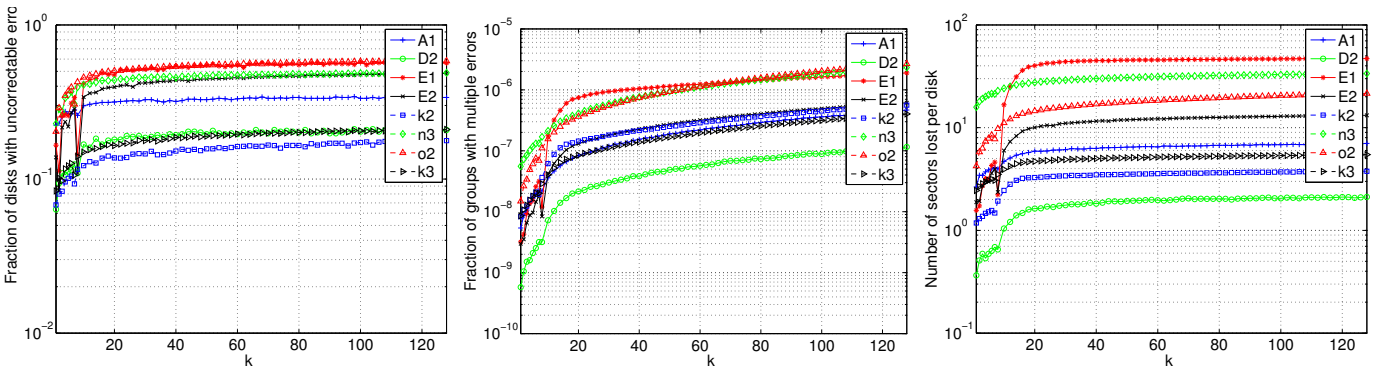

Figure 4: *Evaluation of $k+1$ SPC for different values of k. Fig. 4 (left) shows the fraction of disks with at least one uncorrectable error, i.e. disks that have at least one parity group with multiple errors; Fig. 4 (middle) shows the fraction of parity groups with multiple (and hence uncorrectable) errors; and Fig. 4 (right) shows the average number of sectors with uncorrectable errors per disk (due to multiple errors per parity group)*

ter averaged across all models) sees at least one uncorrectable error (i.e. a parity group with multiple errors). For some models (E-1, E-2, n-3, o-2) nearly 50% of drives see at least one uncorrectable error. On average more than 5 sectors per drive cannot be recovered with intra-disk redundancy. Even under the $1+1$ scheme, which sacrifices 50% of disk space for redundancy, on average 15% of disks have at least one parity group with multiple errors. It is noteworthy that there seems to be little difference in the results between enterprise and nearline drives.

The potential impact of multiple errors in a parity group depends on how close in time these errors occur. If there is ample time between the first and the second error in a group there is a high chance that either a background scrubber or an application access will expose and recover the first error, before the second error occurs. Figure 5 (left) shows the cumulative distribution function of the detection time between the first and the second error in parity groups with multiple errors. We observe that the time between the first two errors is small. More than 90% of errors are discovered within the same scrub interval (2 weeks, i.e. around 2.4×10^6 seconds). We conclude from Figure 5 that multiple errors in a parity group tend to occur at the same time, likely because they have been caused by the same event.

We are also interested in the distribution of the number of errors in groups that have multiple errors. If in most cases most of the sectors in a parity group are erroneous, even stronger protection schemes would not be able to recover those errors. On the other hand, if typically only a small number of sectors (e.g. 2 sectors) are bad, a slightly stronger code would be sufficient to recover those errors. Figure 5 (right) shows a histogram of the number of errors in parity groups with multiple errors for the 8+1 SPC scheme. We observe that across all models the most common case is that of double errors with about 50% of groups having two errors.

The above observations motivate us to look at stronger schemes in the next section.

4.1.2 More complex schemes

This section provides a comparative evaluation of IPC, MDS, CDP and SPC+MDS for varying segment sizes and varying degrees of storage efficiency. Larger segments have the potential for stronger data protection, as they space data and corresponding parity sectors further apart. At the same time larger segments lead to higher I/O overhead, as a write to a data sector requires updating the corresponding parity sector(s), which will require more head movement if the two are spaced further apart.

For CDP, the segment size and the storage efficiency are both determined by its parameter p (which has to be a prime number), while the other schemes are more flexible. In our first experiment we therefore start by varying p and adjusting the parameters of the other schemes to achieve the same m and k (i.e. $k = (p-1)^2$ and $m = 2(p-1)$). The bottom row in Figure 6 shows the results for p ranging from 5 to 23, corresponding to a range of storage efficiency from 66% to 92%, and segment sizes ranging from 24 to 528 sectors. In our second experiment, we keep the storage efficiency constant at 87% (i.e. on average 1 parity segment for 8 data segments), and explore different segment sizes by increasing m and k. The results are shown in the top row of Figure 6. For both experiments we show three different metrics: the fraction of disks with uncorrectable errors (graphs in left column), the average number of uncorrectable sectors per drive (middle column), and the fraction of parity

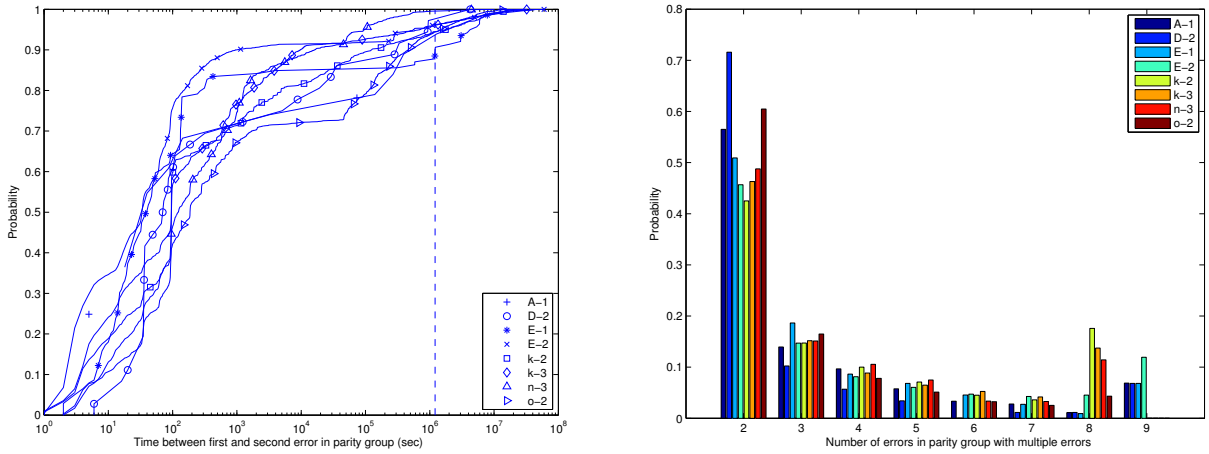

Figure 5: *Distribution of time between the first and second error in 8+1 SPC parity groups with multiple errors (left) and number of errors within a parity group with multiple errors for the case of an 8+1 SPC (right).*

segments with uncorrectable errors (right column).

We observe that all schemes provide clearly superior performance to SPC (for $m = 1$, IPC and MDS reduce to SPC). We also observe that MDS consistently provides the best performance, which might not be surprising as it is the scheme with the highest computational and I/O overheads. Among the remaining schemes CDP performs best, with improvements of an order of magnitude over IPC and SPC+MDS for larger p. SPC+MDS is not as strong, however its improvements of around 25% over simple SPC are impressive given that it applies stronger protection than SPC to only 10% of the total drive.

A surprising result might be the weak performance of IPC compared to MDS or CDP. The original papers [5, 10] proposing the idea of IPC predict the probability of data loss under IPC to be nearly identical to that of MDS. In contrast, we find that MDS (and CDP) consistently outperform IPC. For example, simply moving from an 8+1 to a 16+2 MDS scheme reduces nearly all metrics by 50%. Achieving similar results with an IPC scheme requires at least a 56+7 or 64+8 scheme. For larger segment sizes, MDS and CDP outperform IPC by an order of magnitude.

One might ask why IPC does not perform better. Based on our results in Section 3 we believe there are two reasons. First, the work in [5, 10] assumes that the only correlation between errors is that within an error burst and that different bursts are identically and independently distributed. However, as we saw in Section 3 there are significant correlations between errors that go beyond the correlation within a burst. Second, [5, 10] assumes that the length of error bursts follows a geometric distribution. Instead we found that the distribution of the length of error bursts has long tails (recall Figure 1) and is not fit well by a geometric distribution. As the authors observe in [10] the IPC scheme is sensitive to long tails in the distribution. The above observations underline the importance of using real-world data for modeling errors.

5 Proactive error detection with scrubbing

Scrubbing has been proposed as a mechanism for enhancing data reliability by proactively detecting errors [2, 12, 17]. Several commercial systems, including NetApp's, are making use of a background scrubber. A scrubber periodically reads the entire disk sequentially from the beginning to the end and uses inter-disk redundancy (e.g. provided by RAID) to correct errors. The scrubber runs continuously at a slow rate in the background as to limit the impact on foreground traffic, i.e. for a scrubbing interval s and drive capacity c, a drive is being scrubbed at a rate of c/s. Common scrub intervals are one or two weeks. We refer to a scrubber that works as described above as a *standard periodic scrubber*. In addition to standard periodic scrubbing, we investigate four additional policies.

Localized scrubbing: Given the spatial and temporal locality of LSEs, one idea for improving on standard periodic scrubbing is to take the detection of an error during a scrub interval as an indication that there are likely more errors in the neighborhood of this error. A scrubber could therefore decide upon the detection of an error to immediately scrub also the r sectors that follow the erroneous sector. These neighboring sectors are read at an accelerated rate a, rather than the default rate of c/s.

Accelerated scrubbing: This policy can be viewed as an extreme form of localized scrubbing: Once a bad sector is detected in a scrubbing interval, the entire remain-

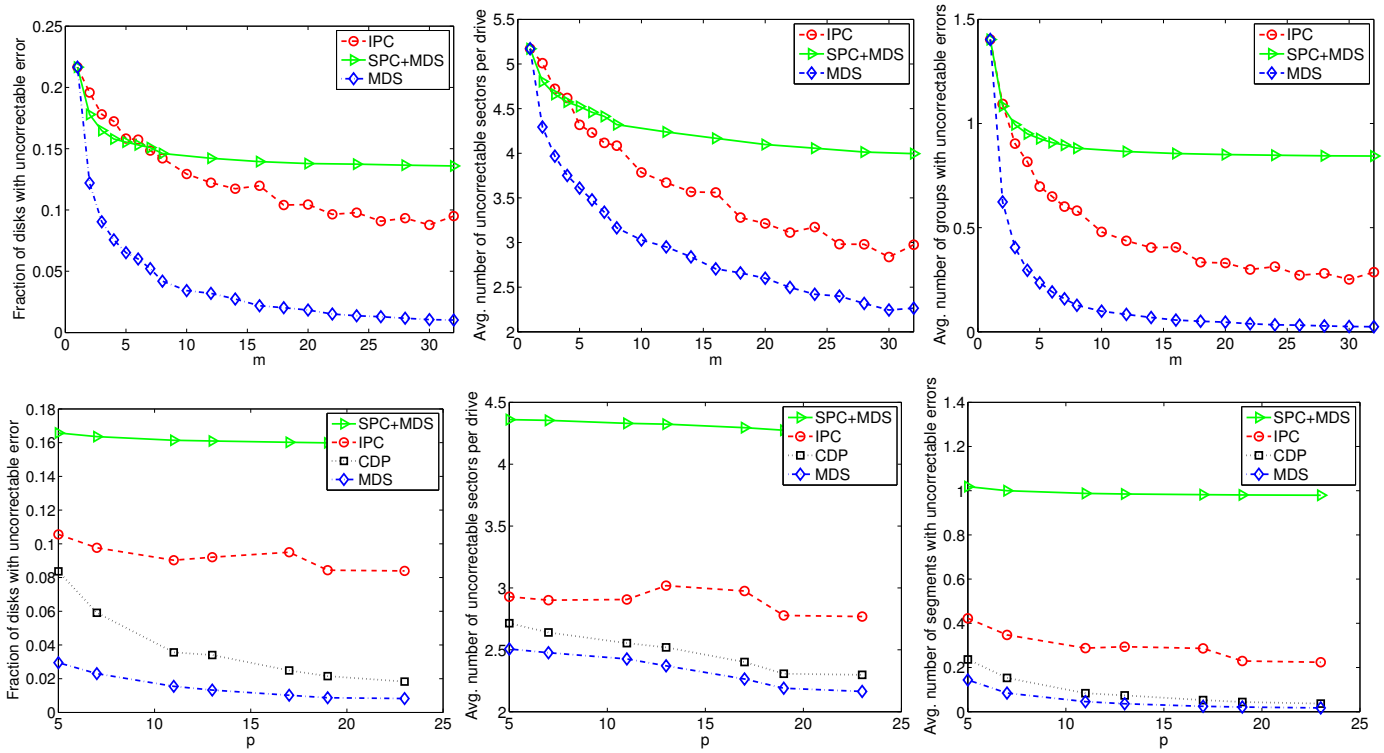

Figure 6: *Comparison of IPC, MDS, SPC+MDS, and CDP under three different metrics: the fraction of disks with at least one uncorrectable error (left), the number of sectors with unrecoverable errors per disk (middle), and the fraction of parity segments that have an unrecoverable error (right). In the top row, we keep the storage efficiency constant by varying m and adjusting $k = 8 \times m$. In the bottom row, we vary the p parameter of CDP and adjust all other policies to have the same m and k values, i.e. $k = (p-1)^2$ and $m = 2(p-1)$.*

der of the drive is scrubbed immediately at an accelerated rate a (rather than the default rate of c/s).

Staggered scrubbing: This policy has been proposed very recently by Oprea et al. [13] and aims to exploit the fact that errors happen in bursts. Rather than sequentially reading the disk from the beginning to the end, the idea is to quickly "probe" different regions of the drive, hoping that if a region of the drive has a burst of errors we will find one in the probe and immediately scrub the entire region. More formally, the drive is divided into r regions each of which is divided into segments of size s. In each scrub interval, the scrubber begins by reading the first segment of each region, then the second segment of each region, and so on. The policy uses the standard scrub rate of c/s and depends on two additional parameters, the segment size s and the number of regions r.

Accelerated staggered scrubbing: A combination of the two previous policies. We scrub segments in the order given by staggered scrubbing. Once we encounter an error in a region we immediately scrub the entire region at an increased scrub rate a (instead of the default c/s).

5.1 Evaluation methodology

Our goal is to evaluate the relative performance of the four different scrubbing policies described above. Any evaluation of scrubbing policies presents two difficulties.

First, the performance of a scrub policy will critically depend on the temporal and spatial properties of errors. While our data contain logical sector numbers and timestamps for each reported LSE, the timestamps correspond to the time when an error was detected, not necessarily the time when it actually happened. While we have no way of knowing the exact time when an error happened we will use three different methods for approximating this time. All methods rely on the fact that we know the time window during which an error must have happened: since the scrub interval on NetApp's systems is two weeks, an error can be latent for at most 2 weeks before it is detected. Hence an error must have happened within 2 weeks before the timestamp in the trace. In addition to running simulations directly on the trace we use the three methods below for approximating timestamps:

Method 1: The strong spatial and temporal locality observed in Section 3 indicate that errors that are

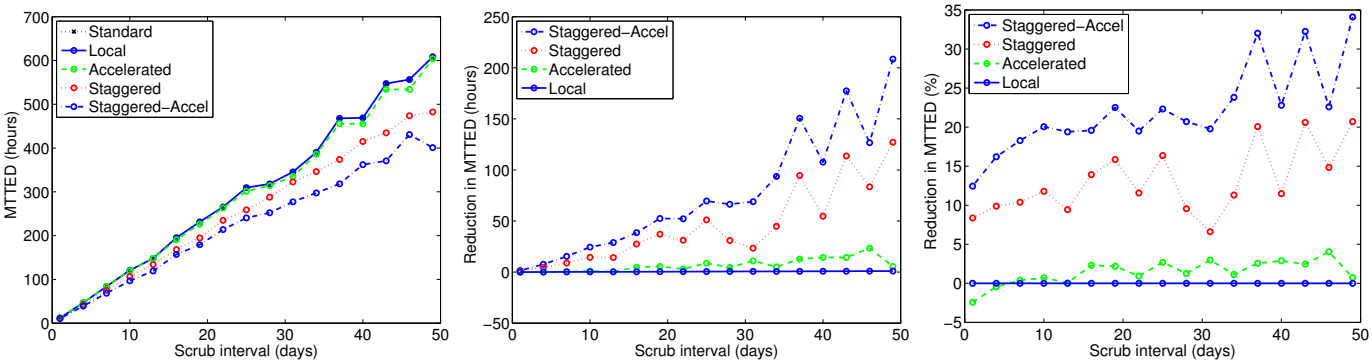

Figure 7: *Comparison of all policies for varying scrub intervals (results averaged across all disk models)*

detected within the same scrub period are likely to be caused by the same error event (e.g. a scratch in the surface or a high-fly write). Method 1 assumes that all errors that happened within a radius of 50MB of each other in the same scrub interval were caused by the same event and assigns all these errors the same timestamp (the timestamp of the error that was detected first).

Method 2: This method goes one step further and assumes that *all* errors that are reported in the same scrub interval happened at the same time (not an unlikely assumption, recall Figure 3) and assigns all of them the timestamp of the first error in the scrub interval.

Method 3: The last method takes an adversary's stance and makes the (unlikely) assumption that all errors in a scrub interval happened completely independently of each other and assigns each error a timestamp that lies randomly in the 2-week interval before the error was detected.

The second difficulty in evaluating scrubbing policies is that there is a possibility that the scrubbing frequency itself affects the rate at which errors happen, i.e. the additional workload created by frequent scrubbing might cause additional errors. After talking to vendors and studying reports [6, 7] on the common error modes leading to LSEs, it seems unlikely that the read frequency in a system (in contrast to the write frequency) would have a major impact on errors. The majority of reported error modes are either directly related to writes (such as high-fly writes) or can happen whenever the disk is spinning, independent of whether data is being read or written (such as thermal asperities, corrosion, and scratches or smears). Nevertheless we are hesitant to assume that the scrub frequency has zero impact on the error rate. Since the goal of our study is not to determine the optimal scrub frequency, but rather to evaluate the relative performance of the different policies, we only compare the performance of different policies under the same scrub frequency. This way, all policies would be equally affected by an increase in errors caused by additional reads.

The main metric we use to evaluate the effectiveness of a scrub policy is the mean time to error detection (MTTED). The MTTED will be a function of the scrub interval since for all policies more frequent scrubs are expected to lead to shorter detection times.

5.2 Comparison of scrub policies

Figure 7 shows a comparison of the four different scrub policies described in the beginning of this section. The graphs, from left to right, show the mean time to error detection (MTTED), the reduction in MTTED (in hours) that each policy provides over standard periodic scrubbing, and the percentage improvement in MTTED over standard periodic scrubbing. We vary the scrub interval from one day to 50 days. The scrub radius in the local policy is set to 128MB. The accelerated scrub rate a for all policies is set to 7000 sectors/sec, which is two times slower than the read performance[2] reported for scrubs in [13]. For the staggered policies we chose a region size of 128MB and a segment size of 1MB (as suggested in [13]). We later also experiment with other parameter choices for the local and the staggered scrub algorithms. When generating the graphs in Figure 7, we took the timestamps verbatim from the trace. In Section 5.2.4 we will discuss how the results change when we use one of the three methods for approximating timestamps, as described in Section 5.1.

5.2.1 Local scrubbing

The performance of the local scrub policy turns out to be disappointing, being virtually identical to that of standard scrubbing. We explain this with the fact that its only potential for improvements lies in getting faster to errors that are within a 128MB radius of a previously detected

[2]The SCSI verify command used in scrubs is faster than a read operation as no data is transferred, so this estimate should be conservative.

error. However, errors within this close neighborhood will also be detected quickly by the standard sequential scrubber (as they are in the immediate neighborhood).

To evaluate the broader potential of local scrubbing, we experimented with different radii, to see whether this yields larger improvements. We find that only for very large radii (on the order of several GB) the results are significant and even then only some of the models show improvements of more than 10%.

5.2.2 Accelerated scrubbing

Similar to local scrubbing, also accelerated scrubbing (without staggering) does not yield substantial improvements. The reasons are likely the same as those for local scrubbing. Once it encounters an error, accelerated scrubbing will find subsequent errors quicker. However, due to spatial locality most of the subsequent errors will be in the close neighborhood of the first and will also be detected soon by standard scrubbing. We conclude that the main weakness of local and accelerated scrubbing is that they only try to minimize the time to find additional errors, once the first error has been found. On the other hand, staggered scrubbing minimizes the time it takes to determine whether there are any errors and in which part of the drive they are.

5.2.3 Staggered scrubbing

We observe that the two staggered policies both provide significant improvements over standard scrubbing for all scrubbing frequencies. For commonly used intervals in the 7-14 day range, improvements in MTTED for these policies range from 30 to 70 hours, corresponding to an improvement of 10–20%. These improvements increase with larger scrubbing intervals. We also note that even simple (non-accelerated) staggered scrubbing yields significantly better performance than both local or accelerated scrubbing, without using any accelerated I/Os.

Encouraged by the good performance of staggered scrubbing, we take a closer look at the impact of the choice of parameters on its effectiveness, in particular the choice of the segment size, as this parameter can greatly affect the overheads associated with staggered scrubbing. From the point of view of minimizing overhead introduced by the scrubber, one would like to choose the segments as large as possible, since the sectors in individual segments are read through fast sequential I/Os, while moving between a large number of small segments requires slow random I/Os. On the other hand if the size of segments becomes extremely large, the effectiveness of staggered scrubbing in detecting errors early will approach that of standard scrubbing (the extreme case of one segment per region leads to a policy identical to standard scrubbing.)

We explore the effect of the segment size for several different region sizes. Interestingly, we find consistently for all region sizes that the segment size has a relatively small effect on performance. As a rough rule of thumb, we observe that scrubbing effectiveness is not negatively affected as long as the segment size is smaller than a quarter to one half of the size of a region. For example, for a region size of 128MB, we find the effectiveness of scrubbing to be identical for segment sizes ranging from 1KB to 32MB. For a segment size of 64MB, the level of improvement that staggered scrubbing offers over standard scrubbing drops by 50%. Oprea [13] reports experimental results showing that for segment sizes of 1MB and up, the I/O overheads of staggered scrubbing are comparable to that of standard scrubbing. That means there is a large range of segment sizes that are practically feasible and also effective in reducing MTTED.

5.2.4 Approximating timestamps

In our simulation results in Figure 7, we assume that the timestamps in our traces denote the actual times when errors happened, rather than the time when they were detected. We also repeated all experiments with the three methods for approximating timestamps described in Section 5.1.

We find that under the two methods that try to make realistic assumptions about the time when errors happened, based on the spatio-temporal correlations we observed in Section 3, the performance improvements of the scrub policies compared to standard scrubbing either stays the same or increases. When following method 1 (all errors detected in the same scrub interval within a 50MB-radius are assigned the same timestamp), the improvements of staggered accelerated scrubbing increase significantly, for some models as much as 50%, while the performance of all other policies stays the same. When following method 2 (all errors within the same scrub interval are assigned the same timestamp) all methods see a slight increase of around 5% in their gains compared to standard scrubbing. When making the (unrealistic) worst case assumption of method 3 that errors are completely uncorrelated in time, the performance improvements of all policies compared to standard scrubbing drop significantly. Local and accelerated scrubbing show no improvements, and the MTTDE reduction of staggered scrubbing and accelerated staggered scrubbing drops to 2–5%.

6 Summary and discussion

The main contributions of this paper are a detailed statistical analysis of field data on latent sector errors and a comparative evaluation of different approaches for protecting against LSEs, including some new schemes that we propose based on our data analysis.

The statistical analysis revealed some interesting properties. We observe that many of the statistical aspects of LSEs are well modeled by power-laws, including the length of error bursts (i.e. a series of contiguous sectors affected by LSEs), the number of good sectors that separate error bursts, and the number of LSEs observed per time. We find that these properties are poorly modeled by the most commonly used distributions, geometric and Poisson. Instead we observe that a Pareto distribution fits the data very well and report the parameters that provide the best fit. We hope this data will be useful for other researchers who do not have access to field data. We find no significant difference in the statistical properties of LSEs in nearline drives versus enterprise class drives.

Some of our statistical observations might also hold some clues as to what mechanisms cause LSEs. For example, we observe that nearly all drives with LSEs, experience all LSEs in their lifetime within the same 2-week period, indicating that for most drives most errors have been caused by the same event (e.g. one scratch), rather than a slow and continuous wear-out of the media.

An immediate implication of the above observation is that both approaches commonly used to model LSEs are unrealistic. The first approach ties LSE arrivals to the workload process, by assuming a certain bit error rate, and assuming that each read or write has the same fixed probability p of causing an LSE. The second approach models LSEs by a separate arrival process, most commonly a Poisson process. Both will result in a much smoother process than the one seen in practice.

In our comparative study of the effectiveness of intra-disk redundancy schemes we find that simple parity check (SPC) schemes still leave a significant fraction of drives (50% for some models) with errors that cannot be recovered by intra-disk redundancy. An observation in our statistical study that a large fraction of errors (for some models 40%) is concentrated in a small area of the drive (the bottom 10% of the logical sector space) leads us to a new scheme that uses stronger codes for only this part of the drive and reduces the number of drives with unrecoverable errors by 30% compared to SPC.

We also evaluate the interleaved-parity check (IPC) scheme [5, 10] that promises reliability close to the powerful maximum distance separable erasure codes (MDS), with much less overhead. Unfortunately, we find IPC's reliability to be significantly weaker than that of MDS. We attribute the discrepancy between our results and those in [5, 10] to the difference between the statistical assumptions (e.g. geometric distribution of error bursts) in [5,10] and the properties of LSEs in the field (long tails in error burst distributions). Finally, we present a new scheme, based on adaptations of the ideas behind row-diagonal parity [4], with significantly lower overheads than MDS, but very similar reliability.

In our analysis of scrubbing policies, we find that a simple policy, staggered scrubbing [13], can improve the mean time to error detection by up to 40%, compared to standard sequential scrubbing. Staggered scrubbing achieves these results just by changing the order in which sectors are scrubbed, without changing the scrub frequency or introducing significant I/O overhead.

Our work opens up a number of avenues for future work. Our long-term goal is to understand how scrubbing and intra-disk redundancy interact with the redundancy provided by RAID, how different redundancy layers should be integrated, and to quantify how different approaches affect the actual mean time to data loss. Answering these questions is not easy, as it will require a complete statistical model that captures spatial and temporal locality, and total disk failures, as well as LSEs.

7 Acknowledgments

We would like to thank the Advanced Development Group at Network Appliances for collecting and sharing the data used in this study. In particular, we thank Lakshmi Bairavasundaram, Garth Goodson, Shankar Pasupathy and Jiri Schindler for answering questions about the data and their systems and for providing feedback on the first drafts of the paper. The first author would like to thank the System Health Group at Google for hosting her during the summer of 2009 and for providing the opportunity to analyze the SMART data collected on their hard disk drives. We also thank Jay Wylie for sharing his insights regarding intra-disk redundancy codes and the anonymous reviewers for their useful feedback. This work was supported in part by an NSERC Discovery grant.

References

[1] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler. An Analysis of Latent Sector Errors in Disk Drives. In *Proc. of SIGMETRICS'07*, 2007.

[2] 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 *Proc. of EuroSys '06*, 2006.

[3] M. Blaum, J. Brady, J. Bruck, and J. Menon. Evenodd: an optimal scheme for tolerating double disk failures in RAID architectures. In *Proc. of ISCA '94*, 1994.

[4] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-diagonal parity for double disk failure correction. In *Proc. of FAST '04*, 2004.

[5] A. Dholakia, E. Eleftheriou, X.-Y. Hu, I. Iliadis, J. Menon, and K. K. Rao. A new intra-disk redundancy scheme for high-reliability RAID storage systems in the presence of unrecoverable errors. *ACM TOS*, 4(1), 2008.

[6] J. G. Elerath. Hard-disk drives: the good, the bad, and the ugly. *Commun. ACM*, 52(6), 2009.

[7] J. G. Elerath and M. Pecht. Enhanced reliability modeling of RAID storage systems. In *Proc. of DSN '07*, 2007.

[8] J. L. Hafner. Weaver codes: highly fault tolerant erasure codes for storage systems. In *Proc. of FAST'05*, 2005.

[9] J. L. Hafner. Hover erasure codes for disk arrays. In *Proc. of DSN '06*, 2006.

[10] I. Iliadis, R. Haas, X.-Y. Hu, and E. Eleftheriou. Disk scrubbing versus intra-disk redundancy for high-reliability raid storage systems. In *SIGMETRICS'08*, 2008.

[11] M. K. Mckusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A Fast File System for UNIX. *ACM TOCS*, 2(3), 1984.

[12] N. Mi, A. Riska, E. Smirni, and E. Riedel. Enhancing data availability in disk drives through background activities. In *Proc. of DSN*, 2008.

[13] A. Oprea and A. Juels. A clean-slate look at disk scrubbing. In *Proc. of FAST '10*, 2010.

[14] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. of SIGMOD*, 1988.

[15] J. S. Plank. The RAID-6 Liberation codes. In *Proc. of FAST'08*, 2008.

[16] V. Prabhakaran, L. N. Bairavasundaram, N. Agrawal, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. IRON File Systems. In *Proc. of SOSP '05*, 2005.

[17] T. Schwarz, Q. Xin, E. L. Miller, D. E. Long, A. Hospodor, and S. Ng. Disk scrubbing in large archival storage systems. In *Proc. of MASCOTS '04*, 2004.

[18] J. J. Wylie and R. Swaminathan. Determining fault tolerance of XOR-based erasure codes efficiently. In *Proc. of DSN'07*, 2007.