Enhancing Data Availability in Disk Drives through Background Activities[*]

Ningfang Mi
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
ningfang@cs.wm.edu

Alma Riska
Seagate Research
1251 Waterfront Place
Pittsburgh, PA 15222
alma.riska@seagate.com

Evgenia Smirni
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
esmirni@cs.wm.edu

Erik Riedel
Seagate Research
1251 Waterfront Place
Pittsburgh, PA 15222
erik.riedel@seagate.com

Abstract

Latent sector errors in disk drives affect only a few data sectors. They occur silently and are detected only when the affected area is accessed again. If a latent error is detected while the storage system is operating under reduced redundancy, i.e., during a RAID rebuild, then data loss may occur. Various features such as scrubbing and intra-disk data redundancy are proposed to detect and/or recover from latent errors and avoid data loss. While such features enhance data availability in the storage system, their execution may cause performance degradation. In this paper, we evaluate the effectiveness of scrubbing and intra-disk data redundancy in improving data availability while the overall goal is to maintain user performance within predefined bounds. We show that by treating them as low priority background activities and scheduling them efficiently during idle times, these features remain performance-wise transparent to the storage system user while still improving data reliability. Detailed trace-driven simulations show that the Mean Time To Data Loss (MTTDL) improves by up to 5 orders of magnitude if these features are implemented independently. By scheduling concurrently both scrubbing and intra-disk parity updates during idle times in disk drives, MTTDL improves by as much as 8 orders of magnitude.

1 Introduction

As digital storage of commercial data and of data for strictly personal use becomes mainstream, high data availability and reliability become imminently critical. Consequently, there are substantial efforts on designing reliable disk-based storage systems by adding redundancy. Data redundancy is traditionally provided using parity locally in the form of disk arrays (e.g., RAIDs) [16], but distributed storage schemes at a broader scale (e.g., the Google File System [7]) enjoy popularity.

Data redundancy protects against entire disk failures as well as failures of disk sectors. Commonly, disk sector errors are known as “latent sector errors”, because they occur silently and are detected only when the affected area on the disk is accessed again [1, 2, 6, 11, 20]. While latent sector errors may be detected when the user tries to access the affected data, it is probable that they are detected during the data rebuild process, because data rebuild accesses the entire disk space in order to restore the redundancy in a system with disk failure(s).

If the storage array is designed to protect from one failure only, such as RAIDs 1 through 5, any latent sector error detected during the data rebuild process causes data loss because there is no data redundancy to protect against the error. Storage systems with two redundancy levels, such as RAID 6 [13] are better protected and may experience data loss because of latent sector errors only if they are detected while two simultaneous failures exist in the array (i.e., a very unlikely event).

To avoid data loss because of latent errors detected during data rebuilds, features exist in the storage system that aim at detecting and recovering from latent sector errors while there is redundancy in the system. Disk *scrubbing* is an error detection technique that detects latent sector errors via background media scans [19]. *Intra*-disk data redundancy is an error recovery technique that adds another level of redundancy in the data by adding parity for sets (segments) of sectors within the same disk [3, 11].

Scrubbing could cause delays to the foreground work because disk operations such as seeks are not preemptive. Furthermore, multiple redundancy levels and intra-disk parity do impose additional work in the storage system when data is modified (e.g., during WRITE operations), because the parity must be updated. Consequently, both scrubbing and intra-disk parity updates should operate as system background processes. If the execution of this additional work competes with regular user traffic, then it may cause unde-

[*]This work was completed in Fall 2006 during N. Mi’s internship at Seagate Research.

sired delays to regular user traffic.

In this paper, we evaluate the effectiveness of scrubbing and intra-disk data redundancy, when these techniques are designed to remain transparent to the storage system user, i.e., keep average user performance degradation within pre-defined targets. For this, we treat both these features as strictly background ones, and schedule them using effective idle time management policies [14]. Throughout our evaluation, we show that our idle time scheduling approach [14], is key to achieving the efficient execution of scrubbing and in particular of the parity updates associated with the intra-disk data redundancy feature.

Detailed trace-driven simulations indicate that, it is possible to effectively detect and recover from latent disk errors even when the system imposes strict limitations on performance degradation of user traffic. The simulation results show that background activities dramatically improve system reliability by achieving several orders of magnitude improvement of its mean time to data loss (MTTDL). Specifically, scrubbing improves the MTTDL by as much as 5 orders of magnitude, while the intra-disk data redundancy scheme evaluated improves the MTTDL by 2 orders of magnitude. We further show that running both scrubbing and intra-disk parity concurrently, utilizes best the entire system. The combination of both features results in significant MTTDL improvements, i.e., as high as 8 orders of magnitude.

This paper is organized as follows. Section 2 presents related work. Section 3 describes background material on modeling of mean time to data lost in storage system. Section 4 describes how to effectively schedule work during idle times in disk drives by taking advantage of the stochastic characteristics of the empirical distribution of disk idle times. Section 5 presents the disk level traces used in our evaluation. Analysis of scrubbing that utilizes idle times is presented in Section 6. Section 7 analyzes background-based intra-disk parity updates. In Section 8 we evaluate performance and data reliability consequences if scrubbing and intra-disk parity updates are simultaneously enabled as background jobs. Conclusions are given in Section 9.

2 Related Work

The metric of interest when it comes to storage system reliability is not the traditional Mean-Time-To-Failure (MTTF) [18], but instead the Mean-Time-To-Data-Loss (MTTDL) [2]. Data loss is caused when additional failures (even of a few data sectors) occur or are detected in a system which has lost its redundancy because of previous failures. While simultaneous failures of multiple disks are rare, disk sector errors may be detected during a data rebuild, i.e., when the system has lost its redundancy, and cause data loss [2].

To avoid data loss, storage systems may be designed with multiple redundancy levels, i.e., RAID 6 [13]. In addition to such solutions, system features such as scrubbing [19] and intra-disk data redundancy [3, 11] represent effective ways to detect and recover from latent sector errors. Recent data show that scrubbing detects as much as 60% of all latent sector errors [1]. These features are preventive in nature but unavoidably introduce more work in the storage system and in the individual disks. To avoid penalizing regular user traffic, any additional work to enhance reliability is completed as a background process during disk or storage system idle times, which is shown to be available in storage systemes [8, 17].

While a myriad of approaches have been proposed to best utilize idle times in order to enhance system performance, reliability, and consistency by exploiting it locally (i.e., within the same system) [10] or remotely (i.e., busy systems may offload part of their work in idle ones) [12], there has been a number of studies that focus solely on how to better manage idle times for scheduling background activities [5, 14]. Methods to adaptively determine how to best exploit disk idle times to schedule long, high-penalty background jobs, such as powering or spinning-down disks, can be found in [4, 9]. On the analytic side, several models have been developed for systems where foreground/background jobs coexist [21].

In this paper, we use the concept of managing idle times proposed in [14], where decision on when to start scheduling a background job is determined by the empirical distribution of the previously monitored idle times. While the study in [14] focuses on the general concept of how to utilize idle times such that the effect on foreground performance is contained within pre-defined bounds, here we focus on customizing these general background job scheduling policies for the specific case of treating scrubbing and intra-disk parity updates as background activities to enhance system reliability. We further study how to best utilize idle times to meet the different needs of an infinite activity such as scrubbing versus a finite one that depends on the specific workload such as intra-disk parity updates, and show dramatic improvements in the mean time to data loss in systems where both features are enabled.

3 Background - MTTDL Estimation

An important reliability metric for storage systems is the Mean-Time-To-Data-Loss (MTTDL). Approximate models for MTTDL as a function of various system parameters are given in [2]. Here, we calculate MTTDL with scrubbing and intra-disk data redundancy using the same models as in [2]. We first provide a quick overview of the MTTDL models presented in [2]. MTTDL is defined based on the following parameters:

- MV, ML: mean interarrival time of visible and latent disk errors, respectively,
- MRV, MRL: mean recovery time from visible and latent errors, respectively,
- MDL: mean detection time of latent sector errors,
- α: temporal locality of errors,
- β_{XY}: spatial locality of errors, where consecutive errors X and Y are either visible (i.e., type V) or latent (i.e., type L).

If no scrubbing is initiated, then MTTDL is given by the following equation:

$$\frac{1}{MTTDL} \approx \frac{\beta_{VV}}{\alpha}\frac{MRV}{MV^2} + \frac{\beta_{LV}}{\alpha}\frac{MRV}{MV \cdot ML} + \frac{1}{ML}. \quad (1)$$

If scrubbing is performed, then the above equation accounts for the average time it takes to detect the error via scrubbing (i.e., MDL) and recover from it (i.e., MRL) as follows:

$$\frac{1}{MTTDL} \approx \frac{\beta_{VV}}{\alpha k^2}\frac{MRV}{ML^2} + \frac{\beta_{LV}}{\alpha k}\frac{MRV}{ML^2} + \frac{\beta_{VL} + k\beta_{LL}}{\alpha k} \cdot \frac{MDL + MRL}{ML^2} \quad (2)$$

where $k = ML/MV$ [2]. The parameter values for Equations (1) and (2) used in the following sections of this paper are given in Table 1 and are those used in [2].

MV	ML	MRV
120,000 hrs	84,972 hrs	1.4 hrs
k	$\alpha, \beta_{VV}, \beta_{LV}, \beta_{VL}$	β_{LL}
1.41	1	0

Table 1. Parameters used for MTTDL estimation.

4 Scheduling Background Activities

Using disk idle times as a separate resource to complete background activities with minimum obstruction to foreground jobs has been the focus of scheduling policies for foreground/background jobs. *Idle waiting* [5] (i.e., delaying scheduling background jobs during idle intervals) is an effective mechanism to reduce foreground performance degradation due to non-preemptive background jobs. An algorithmic approach to estimate how long to idle wait based on the variability of observed idle periods in the system is proposed in [14], where extensive experimentation shows that the efficiency of idle waiting increases as variability of the empirical distribution of idle times increases[1].

Idle waiting combined with an estimation of the number of background jobs to be served within an idle interval, allows meeting foreground performance requirements while serving as many background jobs as possible. The statistical characteristics of idle times can assist in defining how long to idle wait before scheduling background jobs. In [14], the empirical distribution function of idle times is used to determine the length of "idle wait" in the following three background scheduling policies:

body-based: If the variability of idle times is low, then idle wait for a short period. This policy schedules a *few* background jobs in most idle intervals, which results in using the body rather than the tail of the idle times empirical distribution for serving background activities.

tail-based: If the variability of idle times is high, then idle wait for a long period. This policy schedules *many* background jobs in a few idle intervals by using the tail rather than the body of the idle times empirical distribution for serving background activities.

tail+bursty: If burstiness exists in idle times, then it is possible to obtain more accurate prediction about upcoming idle intervals because long idle intervals are "batched" together. After a long idle interval is observed, then it is possible to predict with good accuracy if the next interval is also long, which then allows for more effective scheduling of background activities.

The goal of the body-based policy is to use most of the idle periods in the system and schedule only few background jobs in an idle period. This policy works particularly well for cases with low variability in idle times because all idle intervals are of similar length. In contrast, for idle intervals of high variabilities, the idle waiting time in the tail-based and the tail+bursty policies is much longer, avoiding utilization of most idle periods which results in delaying only few foreground jobs. Although the tail-based policies utilize only few long intervals, the total amount of background work scheduled during those long intervals is yet more when compared to the background work scheduled under the body-based policy. Tail-based policies are effective only if the idle times are highly variable, which implies that very long idle periods are *expected* to eventually occur. In the following sections, we elaborate on how the above policies can be used in the context of scrubbing and intra-disk parity updates to increase MTTDL.

5 Trace Characteristics and Simulation

All policies presented here are evaluated via trace driven simulations, see [17] for a detailed description of the statistical characteristics of the disk drive traces. The selected three disk traces are measured in a personal video recording device (PVR), a software development server, and an e-mail

[1]For more details, we refer readers to the technical report [15].

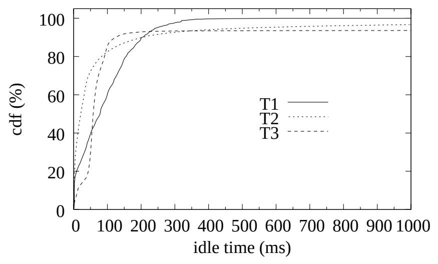

Figure 1. CDF of idle times for traces T1, T2 and T3.

server, which we refer throughout the paper by T1, T2, and T3, respectively. Table 2 gives a summary of the overall characteristics such as request mean interarrival time, request mean service time, utilization, as well as the mean and the coefficient of variation (CV) of idle intervals in the trace. Traces T1, T2 and T3 have 427K, 500K and 362K entries, respectively. They differ from each other in the stochastic characteristics of their idle intervals. For trace T1, idle intervals have a CV close to one, while traces T2 and T3 have higher variability with CVs as high as 6.41 and 3.79, respectively. Consequently, traces T2 and T3 have longer tails than trace T1, see Figure 1. Furthermore, the time series of the observed idle intervals for traces T1 and T2 are not bursty, while the time series of idle intervals for trace T3 is bursty.

Tra-ce	Mean Arrival	Mean Service (ms)	Util (%)	Mean Idle (ms)	CV Idle	Bur-sty
T1	62.85	10.68	17	91.98	0.98	No
T2	96.72	4.20	4.2	236.08	6.41	No
T3	252.29	5.59	2.2	760.84	3.79	Yes

Table 2. Overall characteristics of traces.

The focus here is on the evaluation of scrubbing and parity updates related to intra-disk data redundancy as background activities. Scrubbing is an *infinite* background process because commonly upon completion of one entire disk scan, a new one starts. Parity updates depend on the WRITE user traffic and are considered a *finite* background process. The specific parameters of scrubbing and intra-disk parity update used in our simulations are as follows.

Scrubbing is abstracted as a long background job that is preemptive at the level of a single disk request. Hence, it is assumed that an entire scan of a 40GB disk, i.e., one complete scrubbing, requires 100,000 disk IOs, each scanning approximately 1000 sectors. Disk capacities of 40GB might be conservative given that modern disk drives reach capacities of up to 500GB, but we stress that the analysis presented here is independent of the disk size. One single disk scan request as part of the scrubbing job is assumed to take on the average as much time as a READ disk request. In our simulations, this time is drawn from an exponential distribution with mean 6.0 ms. The time to serve 100,000 disk IOs as part of a single scrubbing corresponds the average scrubbing time.

Parity updates are abstracted as short background jobs. To update the parity of a segment of sectors, the following steps are taken. First, the entire set of sectors should be read, then the parity must be calculated, and finally the new parity is written on the disk. Therefore, each parity update consists of one READ (assumed to take 10 ms on the average) and one WRITE (assumed to take 5 ms on the average), both exponentially distributed. The preemption level of parity updates is at the disk request level. If a parity update is preempted after the READ, then the system maintains no memory of the work done and the update has to restart during another idle period. Parity updates are served in a first-come-first-served (FCFS) fashion.

Scrubbing and intra-disk parity updates processes are scheduled using the three policies outlined in Section 4. In storage systems, a slowdown of the foreground traffic by 5% to 10% is considered transparent to the user. Hence, we set the pre-defined limit on performance degradation to 7%. All three policies degrade the performance of user traffic by restricting the amount of background jobs served. Their efficiency regarding the performance of timely completion of background tasks (i.e., scrubbing or parity updates) depends on the variability of idle times in traces T1, T2, and T3.

6 Disk Scrubbing

Background media scans can be abstracted as an *infinite* background process that detects any possible media errors on disk drives and thus prevents data loss caused by latent sector errors. As a preventive feature, scrubbing is completed in the background and can be conducted by the storage system or the disk drive itself. Based on the system specifications described in Section 5, we evaluate the effectiveness of scrubbing aiming at degrading performance of user traffic by at most 7%.

The goal of scrubbing as a preventive background feature is to improve the MTTDL. The average time of scrubbing allows for MTTDL calculation when scrubbing is not running and when it is running, using Equations (1) and (2), respectively. The mean detection time of sector errors (MDL) in Equation (2) is set to be equal to $0.5\,\times$ average scrubbing time. Moreover, compared to detection times, the recovery times of latent sector errors are insignificant (i.e., MRL $\ll$ MDL). We thus assume MRL ≈ 0 in Equation (2). Table 3 gives the improvements in MTTDL when scrubbing is running over the case when it is not running, i.e., the ratio of the two MTTDLs. The results show that when utilizing the body and the tail of idle times as explained in Section 4, scrubbing dramatically improves reliability while the degradation in the performance of user traffic is limited to 7%. The overall improvement of MTTDL because of scrubbing is 5 orders of magnitude. Differences in the MTTDL improvement between the scheduling policies are between 20% and 40%. The body-based policy achieves better MTTDL improvement for low variable idle times (e.g., trace T1) than the tail-based policy. For T2 that has high viability in idle times, the tail-based policy is superior to the body-based policy. Additionally, the prediction of upcoming idle times further improves the system's reliability for trace T3 whose idle times are bursty.

T1		T2		T3	
body	tail	body	tail	tail	tail+bursty
0.4×10^5	0.3×10^5	0.3×10^5	0.5×10^5	0.4×10^5	0.5×10^5

Table 3. MTTDL improvement via scrubbing.

Table 4 also provides an explanation for the differences in the MTTDL improvement between the three scheduling policies by presenting the number of completed media scans, their average scrubbing time, and the overall system utilization. For lowly variable idle times (e.g., T1), utilizing the body rather than the tail of idle times results in faster scrubbing and better overall system utilization. In particular, scrubbing under the body-based policy is twice faster than under the tail-based policy (see first row of Table 4). Consequently, the faster scrubbing time under the body-based policy yields the superior MTTDL improvements shown in Table 3. For T2 that has highly variable idle times, the tail-based yields faster scrubbing, i.e., at least two orders of magnitude difference than the body-based policy, which results in higher MTTDL improvement. Furthermore, for trace T2, the system is a lot better utilized under the tail-based policy. Finally, if idle times are in addition bursty (i.e., trace T3), then utilizing the tail of idle times and predicting long idle periods performs better than utilizing only the tail of idle times. Utilizing burstiness to benefit scrubbing results in a five-fold improvement in mean scrubbing time. The body-based policy is not evaluated for T3 because the results of T2 establish that the tail rather than the body of idle times should be used if idle times have high variation.

In addition to the average performance presented in Table 4, we also evaluate the distribution of scrubbing time. The distribution is built with a sample space of completed scrubbing as large as 500 by replaying the traces several times. Figure 2 shows the cumulative distribution function (CDF) of scrubbing time for traces T1, T2, and T3. For all three traces, the best performing scheduling policy for scrubbing identified in Table 4 achieves the shortest scrubbing distribution tail. For trace T1, see Figure 2(a), almost 100% of scrubbings have times less than 3831.9 seconds under the body-based policy while a twice larger scrubbing time is achieved for only 1.4% of scrubbings under the tail-

Tra -ce	Policy	Completed Media Scans	Mean Scrubbing Time (s)	System Util (%)
T1	body	6	3,617.8	33.1
	tail	4	6,484.7	26.8
T2	body	4	11,519.6	9.7
	tail	63	726.4	83.1
T3	tail	20	4,476.3	14.3
	tail+ bursty	94	972.9	62.6

Table 4. Scrubbing performance for traces T1, T2, and T3 under body-based, tail-based, and tail+bursty idle time managing policies.

based policy. Similarly for trace T2, see Figure 2(b), the tail of scrubbing time under the tail-based policy is about 7.5 times shorter than under the body-based policy. Exploiting burstiness with the tail+bursty policy, as shown in Figure 2(c), further reduces the tail of the scrubbing time distribution.

7 Intra-disk Parity Updates

Intra-disk data redundancy requires maintaining updated parity that becomes dirty if the corresponding data is modified [3, 11]. This extra amount of work required to maintain updated parity consists of an extra READ and an extra WRITE for each user-issued WRITE. Completing this work instantaneously upon completion of each user-issued WRITE is called *instantaneous parity* (IP) update. Naturally, IP causes degradation in user performance because it is not a preemptable task, but provides the highest level of data reliability.

Here, we show that it is possible to complete parity updates as a background job in a timely fashion, while keeping user performance slowdown within the predefined target 7%. We quantify how the amount of delay in intra-disk parity updates affects data reliability for the three idle scheduling techniques. The effectiveness of the idle scheduling policies are evaluated in comparison to IP updates.

7.1 MTTDL in Data Redundant Drives

The estimation of MTTDL for disks with intra disk redundancy is based on Equation (1). Assuming that latent sector errors are spatially and temporally correlated [1], the improvement in the mean interarrival time of latent sector errors is 0.48×10^2 [3], or equivalently, $ML^{(2)} = 0.48 \times 10^2 \cdot ML^{(1)}$, where $ML^{(1)}$ represents the mean interarrival time of latent errors if there is no intra-disk data redundancy, see Table 1. $ML^{(2)}$ represents the mean interarrival time of latent errors if there is intra-disk data redundancy.

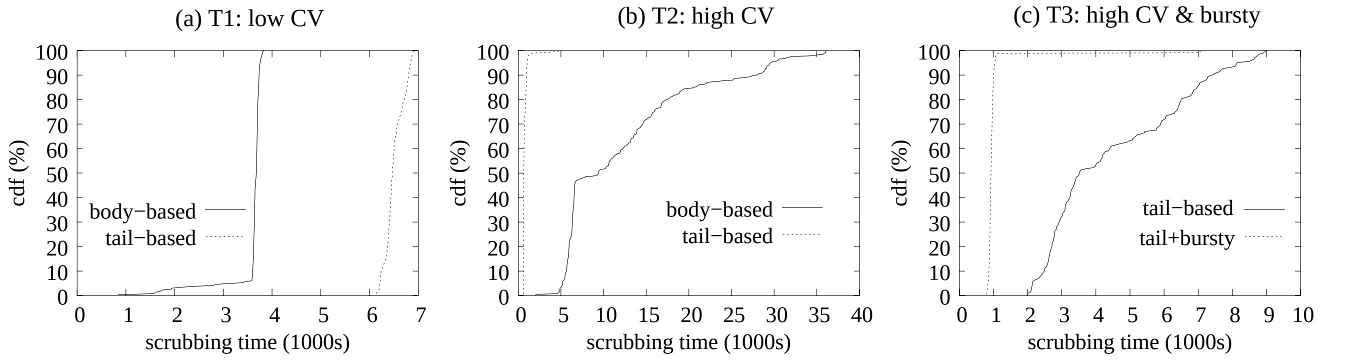

Figure 2. CDF of scrubbing time distribution for traces (a) T1, (b) T2, and (c) T3.

If instantaneous parity (IP) is supported (i.e., parity updates occur without delay), then MTTDL is calculated using Equation (1), where the parameter ML is set as $ML^{(2)}$. If parity updates are delayed, then Equation (1) is modified as follows:

$$\begin{aligned} MTTDL &\approx p \cdot MTTDL_{ML^{(1)}} \\ &\quad +(1-p) \cdot MTTDL_{ML^{(2)}}, \end{aligned} \tag{3}$$

where p represents the probability that the parity is dirty, and $MTTDL_{ML^{(1)}}$ and $MTTDL_{ML^{(2)}}$ are computed using Equation (1) with the parameter ML equal to $ML^{(1)}$ and $ML^{(2)}$, respectively. We assume that if the parity is dirty, then latent errors arrive in intervals of $ML^{(1)}$, and that if parity is updated, then errors arrive in intervals of $ML^{(2)}$. We approximate p as the portion of the disk with dirty parity as follows:

$$\begin{aligned} p &\approx \frac{QL_{Update} \cdot Length_{Parity\ segment}}{Capacity_{Disk}} \\ &= \frac{RT_{Update} \cdot \lambda_{Update} \cdot Length_{Parity\ segment}}{Capacity_{Disk}}, \end{aligned} \tag{4}$$

where QL_{Update} is the average number of dirty parities in the disk, RT_{Update} is the average parity update time, λ_{Update} is the arrival rate of parity updates and $Length_{Parity\ segment}$ is the number of sectors in each parity segment. The performance of the policy to schedule background requests during idle intervals determines RT_{Update} and consequently affects the MTTDL.

In the following, we present results for traces T1 and T2. Traces T2 and T3 yield similar results because both have high variability in idle times and also because for the finite work generated by parity updates exploiting burstiness does not yield any further improvement. The following four metrics are monitored: (a) the MTTDL improvement via intra-disk parity, (b) the ratio of completed parity updates to the total trace WRITE traffic, (c) the average time of parity updates which is the time interval between the completion of a user-issued WRITE operation and the update of the parity, and (d) the overall (foreground + background) system utilization.

7.2 Parity Updates under Trace T1

Assuming that the disk capacity is 40GB, the relative MTTDL improvement estimated for parity updates under trace T1, which has nearly 40% user WRITEs, is given in Table 5. Recall that the relative MTTDL improvement is defined as the difference between MTTDL under the cases with and without intra-disk parity. Here, such an improvement attributed to intra-disk parity is only two orders of magnitude – recall that those attributed to scrubbing are as high as five orders of magnitude. The important result of Table 5 is that there is almost no difference between the MTTDL improvement achieved via instantaneous parity (IP) updates and the delayed parity updates evaluated in this paper, which strongly argues in favor of delayed intra-disk parity. Furthermore, for finite background activities (e.g., parity updates) under trace T1, the tail-based policy achieves slightly better improvement in system reliability than the body-based policy.

	Policy		
Trace	body	tail	IP
T1	0.481×10^2	0.484×10^2	0.484×10^2

Table 5. MTTDL improvement for trace T1 via intra-disk data redundancy, where IP is instantaneous parity update without delaying.

Policy	Completed Ratio (%)	Mean Update Time (s)	System Util (%)
body	38.6	180.6	24.7
tail	41.6	3.3	22.9

Table 6. Parity update performance for trace T1 (low variability).

Table 6 further shows that for this experiment the tail-based scheduling performs better than the body-based policy across all metrics. Most importantly, the tail-based policy updates parities almost by two orders of magnitude faster than the body-based policy. Quick parity update times are particularly desirable because the average parity update time is the metric that affects data reliability. Note that system utilization is higher under the body-based than under the tail-based policy. Under the body-based policy, there are more cases where a user request preempts a parity update. Unfortunately, this results in wasted work. Under the tail-based policy, only long idle intervals are used to update the finite parities which results in only few of them being preempted by foreground traffic.

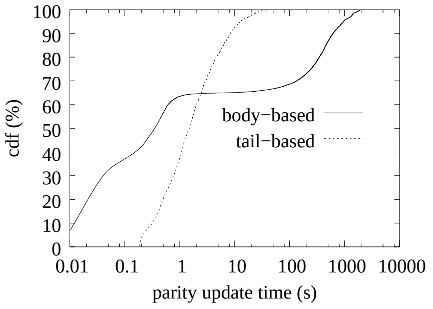

Figure 3. CDF of parity updates time for trace T1 (low variability).

Figure 3 shows the distribution of the parity update times. While about 68% of parity updates under the body-based policy are faster than under the tail-based policy, the tail of parity update times is longer and dominates the average parity update time. This causes a two orders of magnitude advantage for the average tail-based performance, see the mean parity update times in Table 6.

7.3 Parity Updates under Trace T2

User issued WRITE traffic in T2 represents only 1% of the total requests. To experiment with traces with more WRITE traffic, we generate three additional traces that have 10%, 50%, and 90% WRITEs, respectively. These traces are generated based on T2, by probabilistically selecting an entry in the trace to be a READ or a WRITE.

Table 7 presents the relative MTTDL improvement via parity updates under four variants of trace T2 using the body-based and tail-based policies to schedule work in idle times. This table also shows two different performances for the tail-based policy (marked as "tail-S" and "tail-L"). Although both tail-based policies utilize the tail of the idle times, under "tail-S" the idle wait is (approximately 40%) shorter than under "tail-L". Table 7 shows that similar to the results for trace T1 (see Table 5), all scheduling policies achieve two orders of magnitude MTTDL improvement attributed to intra-disk parity. The difference between the MTTDL improvement achieved via instantaneous parity (IP) updates and the delayed parity updates is negligible, especially under the tail-based policy. Recall that IP updates dirty parities instantaneously upon the completion of each user-issued WRITE and is not preemptable. When the amount of parity updates is large (e.g., cases with 50% and 90% WRITEs), the body-based policy obtains the worst MTTDL improvement.

Table 8 shows that since T2 has highly variable idle times, the tail-based policy outperforms the body-based one. For example, the body-based policy performs at least seven times worse than the tail-based policy with respect to the average parity update time. The differences in performance between the body-based and the tail-based policies increase as the amount of parity updates increases. Among the tail-based policies, "tail-L" achieves better update time, while "tail-S" complets more parity updates. The two tail-based policies perform exactly the same when the amount of parity updates is small (e.g., cases with 1% WRITEs). Timely updates are critical for MTTDL, we elaborate more on this later in this section.

Trace	Policy	Parity Update Ratio (%)	Parity Update Time (s)	System Util (%)
T2 (1%)	body	28.40	141.4	4.68
	tail-S	66.40	25.4	4.57
	tail-L	66.40	25.4	4.57
T2 (10%)	body	13.39	7,272.9	5.23
	tail-S	33.20	194.3	5.03
	tail-L	21.88	62.9	4.75
T2 (50%)	body	3.96	9,721.2	5.42
	tail-S	21.57	69.2	6.36
	tail-L	17.44	31.4	5.88
T2 (90%)	body	2.41	8,142.6	5.44
	tail-S	18.36	48.6	7.36
	tail-L	16.90	33.4	7.03

Table 8. Performance of parity updates for trace T2 (high variability) and four different user WRITE traffic, i.e., 1%, 10%, 50% and 90% (numbers in parenthesis indicate the absolute number of user WRITEs).

The overall system utilization in Table 8 is not as high as the 80% utilization level under scrubbing in Table 4 because parity updates represent a finite amount of work. Similarly to the results of trace T1, if the amount of parity updates is small (cases with 1% and 10% WRITEs), then the body-based policy utilizes the system more than the tail-based policy because of the preempted updates. As the amount of parity updates increases, the effect of this phenomenon diminishes.

Figure 4 plots the CDFs of parity update times for all four variants of trace T2. Consistently with results in Table 8, under the body-based policy, the distribution has

Policy	T2			
	1% WRITEs	10% WRITEs	50% WRITEs	90% WRITEs
body	0.484×10^2	0.466×10^2	0.386×10^2	0.351×10^2
tail-S	0.484×10^2	0.483×10^2	0.482×10^2	0.482×10^2
tail-L	0.484×10^2	0.484×10^2	0.483×10^2	0.483×10^2
IP	0.484×10^2	0.484×10^2	0.484×10^2	0.484×10^2

Table 7. MTTDL improvement for trace T2 via intra-disk data redundancy.

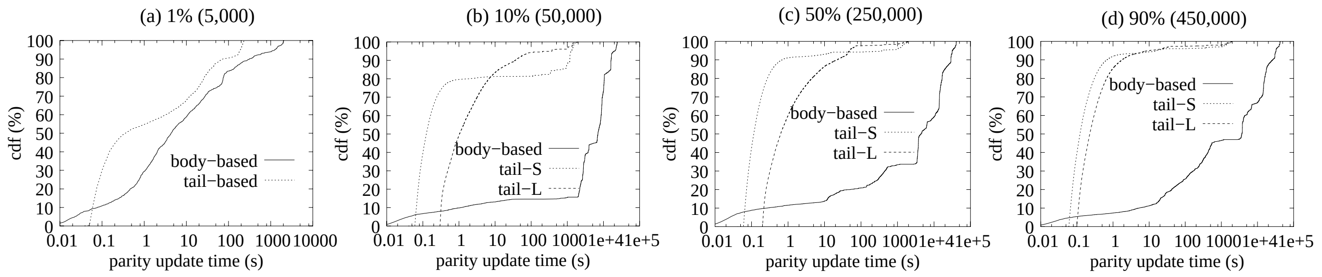

Figure 4. CDF of parity update time for trace T2 (high variability) and four different user WRITE traffic, i.e., 1%, 10%, 50% and 90% (numbers in parenthesis indicate the absolute number of user WRITEs).

longer tail than under the tail-based policy. The "tail-L" variant has the shortest tail indicating that the best average performance comes from the policy that results in a shorter tail of update times. The "tail-L" variant has also the longest idle waiting, which indicates that it uses the smallest number of idle intervals among all policies evaluated, i.e., it waits for the very long idle intervals to arrive. Nevertheless, it results in the shortest mean parity update time and distribution tail. As parity updates increase in number, the differences in the distribution of update times between "tail-S" and "tail-L" decrease.

8 Multi-feature Case: Scrubbing and Intra-disk Parity

Scrubbing and intra-disk parity can be used simultaneously to improve MTTDL. In this section, we evaluate the performance of these two features when running concurrently. Because both features run in the background without any buffer requirement, their queue capacity is assumed to be infinite. Recall that scrubbing generates *infinite* work while parity updates require *finite* work. Here, we evaluate a scenario when parity updates have higher priority than scrubbing, i.e., scrubbing is scheduled only if there is no parity update waiting. As in previous sections, the performance degradation of user traffic is kept below the preset 7% threshold.

8.1 MTTDL in Data Redundant Drives

We use Equation (3) to estimate the MTTDL improvement when both scrubbing and intra-disk parity are enabled. Differently from the MTTDL estimation in Section 7, the $MTTDL_{ML^{(1)}}$ and $MTTDL_{ML^{(2)}}$ in Equation (3) are computed using Equation (2). The average time for a complete disk scrubbing when it runs concurrently with parity updates is used in Equation (2) to estimate both $MTTDL_{ML^{(1)}}$ and $MTTDL_{ML^{(2)}}$, i.e., $ML^{(1)} = 0.5 \times$ average scrubbing time and $ML^{(2)} = 0.48 \times 10^2 \cdot ML^{(1)}$. As the same as in Section 6, we assume MRL ≈ 0 in Equation (2). The parameter p in Equation (3) is estimated using Equation (4) and the average parity update time when it runs concurrently with scrubbing.

8.2 Results for Trace T1

Here we present results for trace T1 which is characterized by idle periods with low variability. For this trace, scrubbing performs better with the body-based policy while parity updates are done more efficiently under the tail-based policy. Here, in addition to the body-based and the tail-based policies, we also evaluate a "hybrid" scheduling policy that schedules scrubbing work via the body-based policy and parity updates via the tail-based policy. This policy is called "body+tail" policy.

Table 9 presents the MTTDL improvement when scrubbing and intra-disk parity co-exist under the three scheduling policies. As expected, for T1 that has idle times of low variablity, the body+tail policy achieves the best improvement in MTTDL. Most importantly, the new combined policy gives 8 orders of magnitude improvements in MTTDL. Comparing to running scrubbing and parity updates individually, using both features results in dramatically higher improvements while keeping the degradation of the fore-

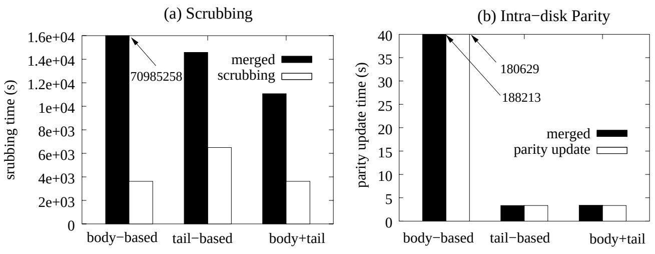

Figure 5. Average time for (a) an entire scrubbing, (b) parity updates for trace T1 (low variability).

ground performance within pre-defined limits.

Trace	T1		
Policy	body	tail	body+tail
	$0.2{\times}10^8$	$0.6{\times}10^8$	$0.7{\times}10^8$

Table 9. MTTDL improvement for trace T1 (low variability) via scrubbing and intra-disk parity.The body+tail policy schedules scrubbing via the body-based policy and parity-updates via the tail-based policy.

Figure 5(a) presents the average time for a complete scrubbing when run individually and when merged together with parity updates. If the body-based policy is used to schedule both types of background jobs, performance degradation on scrubbing is significant. With the body+tail variation, each background activity (i.e., scrubbing or parity update) is scheduled using the policy under which it performs best. Parity updates, because they have higher priority than scrubbing, are not penalized as much as scrubbing (see Figure 5(b)). Furthermore, parity updates perform significantly better if they are scheduled using the tail-based policy, independently of how scrubbing is scheduled.

Figure 6 presents system utilization for trace T1. Results are in agreement with those shown in Figure 5: the body+tail policy utilizes best the entire system providing room for both scrubbing and parity updates to provide significant performance improvements.

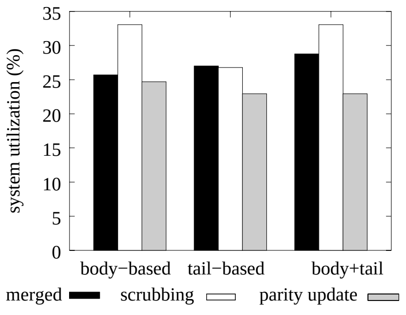

Figure 6. Overall system utilization for trace T1 (low variability) via scrubbing and intra-disk parity.

8.3 Results for Trace T2

Now, we present the results for T2 with high variable idle times. For this trace, both scrubbing and parity updates individually perform better using the tail-based policy. Table 10 gives the MTTDL improvement attributed to scrubbing and intra-disk parity under this policy only. For the four variants of trace T2, the background activities dramatically improve the system reliability by improving its MTTDL by as high as 8 orders of magnitude. Consistently with the results shown in Table 9, there are gains of at least 3 orders of magnitude in MTTDL.

Policy	T2			
	1% WRITEs	10% WRITEs	50% WRITEs	90% WRITEs
tail	$1.12\ {\times}10^8$	$1.09{\times}10^8$	$1.07\ {\times}10^8$	$1.04{\times}10^8$

Table 10. MTTDL improvement for trace T2 (high variability) via scrubbing and intra-disk parity.

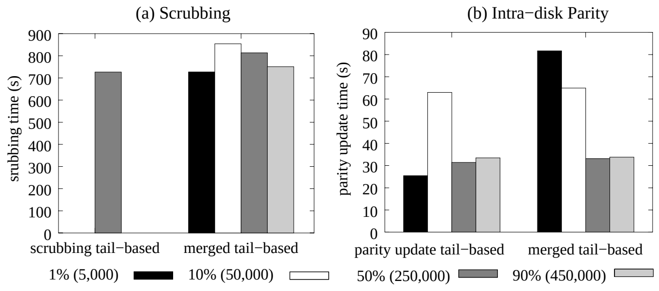

Figure 7. Average (a) scrubbing and (b) parity update times when running individually and together.

Figures 7(a) and 7(b) further give the average scrubbing and parity update times. For comparison, in each plot the results of only disk scrubbing and only intra-disk parity are also included, see left set of columns in Figures 7(a) and 7(b), respectively. For the case of scrubbing, all variants of trace T2 perform the same because scrubbing is workload independent.

Although scrubbing has lower priority than intra-disk parity update, enabling it concurrently with parity updates does not affect its performance considerably (i.e., only 10% WRITEs in the worst case). Similarly, parity updates see minimal change in their performance because they are processes of higher priority than scrubbing. The only exception is the case with the smallest amount of parity updates (i.e., only 1% user WRITEs). As discussed in Section 7, the effect of parity updates in user traffic performance is almost zero for this case and parity update times are the smallest. However, adding the infinite scrubbing work degrades parity update performance by as much as 3 times.

Figure 8 shows overall system utilization, which is dominated by the work done for scrubbing. Because the work related to parity updates is small, its completion barely adds to the system utilization. It is scrubbing with its infinite amount of work that keeps the system continuously utilized.

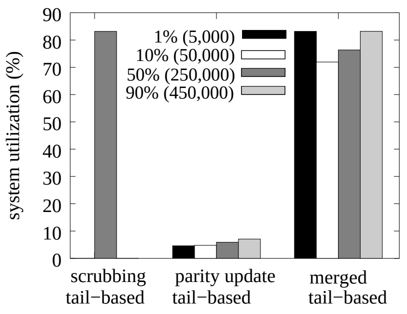

Figure 8. Overall system utilization under scrubbing and parity updates when they run individually and together.

9 Conclusions

In this paper, we evaluate the effectiveness of data loss prevention techniques, such as disk scrubbing and intra-disk data redundancy, when their execution should not affect user performance more than pre-defined bounds. Our trace driven evaluation indicates that treating these features as strictly background features and scheduling them during idle times, guided by advance idleness management techniques, do achieve the goal of maintaining user performance degradation at minimum, while significantly improving the storage system Mean Time To Data Loss - MTTDL. Specifically, scrubbing improves MTTDL by up to five orders of magnitude and intra-disk data redundancy improves MTTDL by up to two orders of magnitude. However, running concurrently both features yields further gains with respect to MTTDL, i.e., up to eight orders of magnitude, without violating the user performance constraints. These results indicate that these two features complement each-other and significantly improve data availability and reliability in the storage system while remaining strictly low priority features.

References

[1] Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler. An analysis of latent sector errors in disk drives. *SIGMETRICS Perform. Eval. Rev.*, 35(1):289–300, 2007.

[2] M. Baker, M. Shah, D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, T. J. Giuli, and P. Bungale. A fresh look at the reliability of long-term digital storage. In *Proceedings of European Systems Conference (EuroSys)*, pages 221–234, April 2006.

[3] A. Dholakia, E. Eleftheriou, X. Y. Hu, I. Iliadis, J. Menon, and K. K. Rao. Analysis of a new intra-disk redundancy scheme for high-reliability RAID storage systems in the presence of unrecoverable errors. Technical report, RZ3652, IBM Reasearch, 2006.

[4] Fred Douglis, P. Krishnan, and Brian N. Bershad. Adaptive disk spin-down policies for mobile computers. In *Proceedings of the 2nd USENIX Symposium on Mobile and Location-Independent Computing*, pages 121–137, 1995.

[5] L. Eggert and J. D. Touch. Idletime scheduling with preemption intervals. In *Proceedings of the International Symposium on Operating Systems Principles (SOSP)*, pages 249–262, October 2005.

[6] Jon G. Elerath and Michael Pecht. Enhanced reliability modeling of raid storage systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 175–184, 2007.

[7] S. Ghemawat, H.Gobioff, and S. Leung. The Google file system. In *Proceedings of ACM Symposiom on Operating Systems Principles*, pages 29–43, 2003.

[8] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes. Idleness is not sloth. In *Proceedings of the Winter'95 USENIX Conference*, pages 201–222, New Orleans, LA, January 1995.

[9] David P. Helmbold, Darrell D. E. Long, Tracey L. Sconyers, and Bruce Sherrod. Adaptive disk spin-down for mobile computers. *Mobile Networks and Applications*, 5(4):285–297, 2000.

[10] Hai Huang, Wanda Hung, and Kang G. Shin. FS2: dynamic data replication in free disk space for improving disk performance and energy consumption. In *Proceedings of the twentieth ACM symposium on Operating systems principles (SOSP)*, pages 263–276, 2005.

[11] Gordon F. Hughes and Joseph F. Murray. Reliability and security of RAID storage systems and D2D archives using SATA disk drives. *ACM Transactions on Storage*, 1(1):95–107, 2005.

[12] Virginia Mary Lo, Daniel Zappala, Dayi Zhou, Yuhong Liu, and Shanyu Zhao. Cluster computing on the fly: P2P scheduling of idle cycles in the internet. In *the 3rd International Workshop on Peer-to-Peer Systmes (IPTPS)*, pages 227–236, 2004.

[13] C. Lueth. RAID-DP: Network appliance implementation of RAID double parity for data protection. Technical report, Technical Report No. 3298, Network Appliance Inc, 2004.

[14] N. Mi, A. Riska, Q. Zhang, E. Smirni, and E. Riedel. Efficient management of idleness in systems. *Proceedings of SIGMETRICS*, 35(1):371–372, 2007.

[15] N. Mi, A. Riska, Q. Zhang, E. Smirni, and E. Riedel. Efficient utilization of idle times. Technical report, WM-CS-2008-01, Department of Computer Science, William and Mary, 2008.

[16] D. A. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proceedings of the 1988 ACM SIGMOD Conference*, pages 109–116. ACM Press, 1988.

[17] A. Riska and E. Riedel. Disk drive level workload characterization. In *Proceedings of the USENIX Annual Technical Conference*, pages 97–103, May 2006.

[18] Bianca Schroeder and Garth A. Gibson. Disk failures in the real world: what does an mttf of 1,000,000 hours mean to you? In *Proceedings of the 5th conference on USENIX Conference on File and Storage Technologies (FAST)*, pages 1–1, 2007.

[19] T. J. E. Schwarz, Q. Xin, E. L. Miller, D. D. E. Long, A. Hospodor, and S. Ng. Disk scrubbing in large archival storage systems. In *Proceedings of the International Symposium on Modeling and Simulation of Computer and Communications Systems (MASCOTS)*, pages 409–418, 2004.

[20] S. Shah and J. G. Elerath. Reliability analysis of disk drive failure mechanism. In *Proceedings of 2005 Annual Reliability and Maintainability Symposium*, pages 226–231. IEEE, January 2005.

[21] Qi. Zhang, N. Mi, E. Smirni, A. Riska, and E. Riedel. Evaluating the performability of systems with background jobs. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, pages 495–504, 2006.