Enhancing Data Availability through Background Activities

Ningfang Mi
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
ningfang@cs.wm.edu
Phone: 1-757-221-3484

Alma Riska
Seagate Research
1251 Waterfront Place
Pittsburgh, PA 15222
alma.riska@seagate.com
Phone: 1-412-918-7020

Evgenia Smirni
Computer Science Dept.
College of William and Mary
Williamsburg, VA 23187
esmirni@cs.wm.edu
Phone: 1-757-221-3580

Erik Riedel
Seagate Research
1251 Waterfront Place
Pittsburgh, PA 15222
erik.riedel@seagate.com
Phone: 1-412-918-7025

Abstract

Latent sector errors in disk drives affect only a few data sectors and are often not detected till the affected data is accessed again. They may cause data loss if the storage system is operating under reduced redundancy, because of previous failures. In this paper, we evaluate effectiveness of two known techniques to detect and/or recover from latent sector errors, namely scrubbing and intra-disk data redundancy. These two techniques are treated as background activities that complete without affecting the otherwise normal operation of the storage system. We focus on how disk idle times can be managed to effectively complete these background tasks without affecting foreground task performance, while reducing the window of vulnerability for data loss. We show via detailed trace-driven simulations that scheduling policies for background jobs that are based on careful monitoring of the stochastic characteristics of idle times in disk drive, have a minimal effect on foreground task performance while dramatically improving storage system reliability.

Keywords: *Foreground/background jobs; storage systems; idle periods.*

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. Disk drives are a versatile permanent storage that offers a wide range of capacities and efficient data access times. Consequently, there are substantial efforts that focus on designing of reliable disk-based storage systems by adding redundancy. Data redundancy traditionally is provided using parity locally in the form of disk arrays (i.e., RAIDs) [17], but distributed storage schemes at a broader scale (e.g., the Google File System [7]) enjoy popularity as the platform of choice for large Internet service centers.

Upon a single disk failure, storage systems are designed to restore the lost data using redundant information. The data restoring process is not instantaneous and, for large capacity disks, can be up to 36 hours. During data restoration the storage system operates with reduced redundancy and any additional failure causes data loss. Although a second entire disk failure during this period is less likely, data loss may occur even if a few disk sectors are not accessible. Disk sector errors, often related to localized media failures, are more frequent than entire disk failures and are known as "latent sector errors" because they are detected only when the affected area on the disk is accessed and not when they truly occur [20, 3].

Data loss [3, 11] due to latent sector errors is addressed by supporting multiple redundancy levels in RAID arrays, e.g., RAID 6 [14] protects from two failures while RAIDs 1 through 5 protect from one failure only. In general, there are two prevalent strategies for protecting data from latent sector errors: first detect them and second recover lost data using the inherent redundancy of the storage system. Disk *scrubbing* is an error detection technique that aims at detecting latent sector errors via background media scan and *before* the affected data is accessed by the user or before any other disk failure [19]. *Intra*-disk data redundancy is used as an error recovery technique by adding parity for sets (segments) of sectors within the same disk [4, 11] which is effective in multiple- and single-disk storage systems.

However, scrubbing could cause delays to the foreground work because disk operations such as seeks are not preemptive. Multiple redundancy levels and intra-disk parity do impose additional work in the storage system when data is modified (i.e., during WRITE operations) because the parity need to be updated. Consecutively, both scrubbing and intra-disk parity updates can operate as system background processes, because if the execution of this additional work competes with regular user traffic, it may cause additional undesired delays.

In this paper, we evaluate the impact that the idle time management policies have on performance of the above background activities and consecutively on data reliability, when they are constraint by the degradation on foreground performance. Such idle time management is in the same spirit as the techniques proposed in [15], where idle times stochastic characteristics (i.e., variability and burstiness) guide background work scheduling. Detailed trace-driven simulation indicates that detection and recovery from latent disk errors can be effective even when the system imposes a strict limitation on performance degradation of user traffic (i.e., maximum set at 7%).

First, the performance of the proposed scrubbing and intra-disk parity updates are evaluated individually. The simulation results show that for background activities with infinite amount of work such as scrubbing, scheduling should follow idle times characteristics, especially variability and burstiness. For background activities with finite amount of work such as the parity updates, a general rule of thumb is that using the tail of the idle times rather than the body yields faster completion times and shorter response time tails. The background activities dramatically improve system's reliability by improving its mean time to data loss. Second, we present how the concurrent use of both scrubbing and intra-disk parity can result in significant reliability improvement which is higher than the linear combination of their individual improvements.

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 utilizing 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 dle 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) anymore but Mean-Time-To-Data-Loss (MTTDL) [3]. Data loss commonly is caused by detection of latent sector errors during the time that a storage system has reduced redundancy because of a previous disk failure [20].

The straightforward approach to improving data reliability is to add multiple redundancy levels in a storage system as in RAID 6 [14]. Nevertheless, system features such as scrubbing [19] and intra-disk parity [4, 11] are shown critical for detecting and recovering from latent sector errors. Such features are preventive in nature but unavoidably introduce more work in the storage system and in individual disks. To avoid penalizing regular user traffic, any additional work to enhance reliability is completed as a background process, i.e., during disk or storage system idle times, especially given that disks are often underutilized [8, 18].

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, 1, 2], or remotely (i.e., busy systems may offload part of their work in idle ones) [12, 13], a number of studies have focused solely on how to better manage idle times for scheduling background activities [6, 15]. 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 [5, 9]. On the analytic side, several models have been developed for systems where foreground/background jobs coexist [23], including vacation models [16, 22] and queueing models of cycle stealing [21].

In this paper, we use the concept on managing idle times proposed in [15], where decision on when to start scheduling a background job is determined by the empirical distribution of the previously monitored idle times. While [15] focuses on the general concept of how to utilize idle times such that the effect on foreground performance is contained, here we focus on customizing these general techniques for the specific case of scheduling scrubbing and intra-disk parity updates as background activities that enhance system reliability. We further study how to best utilize idle times to meet the different needs of continuous (i.e., 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

Latent sector errors rather than total disk failures cause loss of data but not necessary result in storage system failure. Consequently, an important reliability metric for storage systems is the Mean-Time-To-Data-Loss (MTTDL). Approximate models for the MTTDL as a function of various system parameters are given in [3]. Here, we calculate MTTDL of systems with scrubbing and intra-disk data redundancy using the same models as in [3]. For details on the models, we direct the interested reader in [3]. Here, we only provide a quick overview as follows. The model defines MTTDL in terms of the following parameters:

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.

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: errors temporal locality parameter,

β_{XY}, errors spatial locality parameters, 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 is defined in [3] as $k = ML/MV$. The parameter values for Equations (1) and (2) used in [3] and in the following sections of our paper are given in Table 1.

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 [15, 6]. *Idle waiting* [6] (i.e., delaying scheduling background jobs during idle intervals) is an effective mechanism to reduce the effect that non-preemptive background jobs have in foreground performance. An algorithmic approach to estimate how long to idle wait based on the variability of observed idle periods in the system is proposed in [15] and extensive experimentation shows that the efficiency of idle waiting increases as variability of the empirical distribution of idle times increases.

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. Accurate estimation of the amount of background work to be served proves also critical to the effectiveness of scheduling idle times for background activities [15]. The statistical characteristics of idle times can assist in defining how long to idle wait before scheduling background jobs as follows:

body-based: If the variability of idle times is low, then idle waiting for a short period (appropriately defined by the idle time distribution) and scheduling of *a few* background jobs in most idle intervals results in using the body rather than the tail of the idle times empirical distribution.

tail-based: If the variability of idle times is high, then idle waiting for a long period (appropriately defined by the idle time distribution) and scheduling *many* background jobs in a few idle intervals results in using the tail rather than the body of the idle times empirical distribution.

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.

Figure 1 presents the high level idea of the body-based and tail-based idle time scheduling policies. In each plot of the figure, a horizontal line represents a single idle interval of a specific length. In every idle interval, the scheduler idle waits for a time period, whose length is defined by idle times distribution. Once the idle-wait has elapsed (marked with the first dotted vertical line), the estimated background work is scheduled (marked bold in each horizontal line). The estimated background work is either completed (marked with the second dotted vertical line) or preempted by foreground jobs. Cases when background jobs are preempted by foreground ones are the ones where the horizontal line falls completely on the left of the second dotted vertical line. Similarly, cases where the second dotted vertical line in each plot intersects with a horizontal line indicate that the estimated background work has completed without affecting foreground jobs and the system returns to idle.

Figure 1(a) shows that the goal of the body-based policy is to use most of the idle periods in the system and schedule

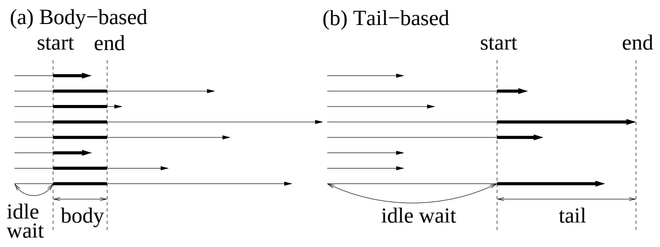

Figure 1. Utilizing (a) the body and (b) the tail of idle times.

Tra-ce	Mean Arrival	Mean Service	Util (%)	Mean Idle	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 used in our evaluation. The measurement unit is ms.

only few background jobs in idle periods used. This policy works particularly well for cases with low variability in idle times because all idle intervals are of similar length. In contrast, for the tail-based and the tail+bursty-based policies, the length of the idle wait is much longer avoiding utilization of most idle periods which results in delay for only few foreground jobs. Because the tail-based policies utilize only few long intervals, the total amount of background work scheduled during those long intervals is 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 for scheduling background tasks can be used in the context of scrubbing and intra-disk parity updates to increase the mean time to data loss (MTTDL).

5 Trace Characteristics and Simulation

All policies presented here are evaluated via trace driven simulation. All simulations are driven by disk drive traces, see [18] for a detailed description of the statistical characteristics of the selected ones. We selected three disk traces that were measured in a personal video recording device (PVR), a software development server, and an e-mail server, which we refer throughout the paper by T1, T2, 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, T3, have 427K, 500K, and 362K entries, respectively. They differ from each other in the characteristics of their idle intervals. For trace T1, idle intervals have a C.V. close to one, while traces T2 and T3 have higher variability with C.V.s as high as 6.41 and 3.79, respectively. 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.

The focus of this paper is the evaluation of two background activities, namely scrubbing and parity updates related to intra-disk data redundancy. Scrubbing is an *infinite* background process because upon completion of one entire disk scan, commonly a new one starts. The parity updates depend on the WRITE user traffic and is considered *finite* background feature. Table 3 gives the specific parameters of scrubbing and intra-disk parity update used in our simulations.

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 completed scrubbing, requires 100,000 disk IOs each scanning approximately 1000 sectors. Assuming disk capacities of 40GB might be conservative given that modern disk drives reach capacities of up to 500GB. Nonetheless, the analysis presented in this paper still holds for larger disks as well. One single disk scan request as part of the scrubbing job is assumed to take in average as much time as a READ disk request. In our simulation, this is drawn from an Exponential distribution with mean 6.0 ms (similarly to the mean service time of traces in Table 2). 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 in average 10 ms) and one WRITE (assumed to take in average 5 ms), 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 again during another idle period. Parity updates are served in a FCFS fashion.

Scrubbing and intra-disk parity update processes are scheduled using the three policies outlined in Section 4. All three policies degrade the performance of user traffic up to 7% (this is a pre-set system parameter) 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. The following

Trace	Scrubbing		Intra-disk Parity Update			
	Short BG Number	Short BG Mean Service	Short BG Number	Read BG Mean Service	Write BG Mean Service	Write Portion
T1	100,000	6.0	2	10.0	5.0	40 %
T2	100,000	6.0	2	10.0	5.0	1%; 10%; 50%; 90%
T3	100,000	6.0	2	10.0	5.0	50%

Table 3. Background activities characteristics. The unit of measurement is ms.

Tra-ce	Policy	Completed Scrubbing	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-based idle time managing policies.

T1		T2		T3	
body	tail	body	tail	tail	tail+bursty
4×10^4	3×10^4	3×10^4	5×10^4	4×10^4	5×10^4

Table 5. MTTDL improvement via scrubbing.

sections further elaborate on policy sensitivity with respect to idle time variability.

6 Infinite Background Activities: Scrubbing

Background media scans can be abstracted as an *infinite* background process that detects any possible media errors on disk drives and thus prevents any data loss caused by the latent sector errors. As a preventive feature, scrubbing is completed in 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%.

Table 4 presents the number of completed media scans, their average scrubbing time, and the overall system utilization for the three traces of Table 2, when utilizing the body and the tail, of idle times as explained in Section 4. Consistently with results reported in [15], for lowly variable idle times (i.e., trace 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). For T2 that has highly variable idle times, the tail-based rather than the body-based policy yields faster scrubbing and better system utilization (i.e., at least an order of magnitude difference, see second row of Table 4). 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 scheduling 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 tail rather than body of idle times should be utilized if idle times have high CV.

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. However, the differences between the scrubbing scheduling policies are more drastic when it comes to the distributions than the average values. For example, for trace T1 (see Figure 2(a)), almost 100% of scrubbings have scrubbing times less than 3831.9 seconds under the body-based policy while a twice as large scrubbing time is achieved only for 1.4% of scrubbings under the tail-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-based policy, as shown in Figure 2(c), further reduces the tail of scrubbing time distribution.

The goal of scrubbing as a preventive background feature is to improve the MTTDL. The average time of scrubbing, given in Table 4, 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

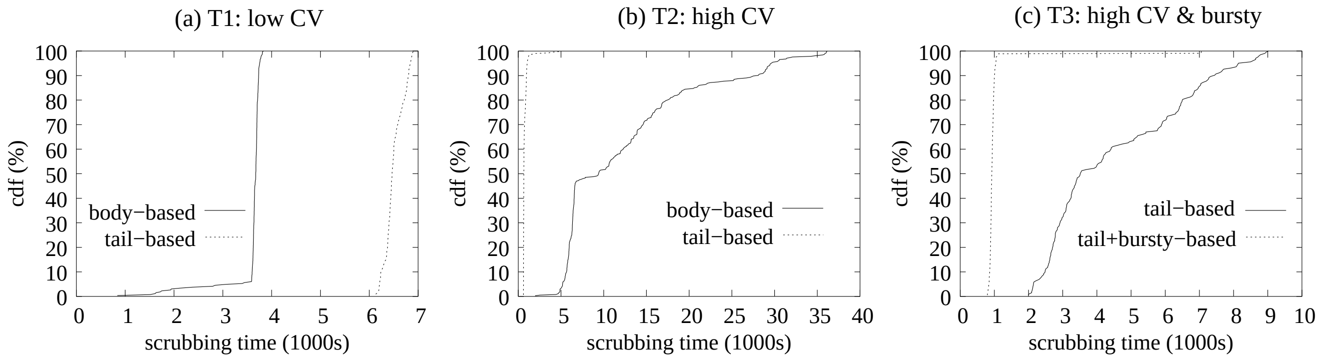

Figure 2. CDF of scrubbing time distribution for traces (a) T1, (b) T2, and (c) T3.

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 5 gives the improvements in MTTDL when scrubbing is running over the case when it is not running. The overall improvement of MTTDL because of scrubbing is 4 orders of magnitude. The differences in the MTTDL improvement between the scheduling policies that are used to manage the idle times are between 20% and 40%.

7 Finite Background Activity: Intra-disk Parity Update

Intra-disk data redundancy requires maintaining updated parity that becomes dirty if the corresponding data is modified [4, 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 but provides the highest level of data reliability.

Here, we show that it is possible to complete the parity updates as a background job scheduled in idle intervals in a timely fashion while keeping user performance slowdown less than a predefined target. In the experiments presented here acceptable user slowdown is set to 7% only. Delays in parity updates reduce the effect of intra-disk parity on data reliability. We quantify how delayed intra-disk parity affects data reliability for the three idle scheduling techniques of Section 4.

We present results for traces T1 and T2. Traces T2 and T3 yield similar results because both have high variability in idle times and because for the finite work generated by parity updates exploiting burstiness does not yield any further improvement. The following three metrics are monitored: (a) the ratio of completed parity updates to the total trace WRITE traffic, (b) 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 (c) the overall (foreground + background) system utilization.

Policy	Completed Ratio (%)	Mean Update Time (ms)	System Util (%)
body	38.6	180,629.0	24.7
tail	41.6	3,321.0	22.9

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

7.1 Parity Updates under Trace T1

Table 6 gives the parity update results under the body-based and tail-based idle time scheduling policies. Trace T1 has nearly 40% user WRITEs. Different from the behavior under infinite background activities (see Section 6), the tail-based rather than the body-based idle time scheduling performs best overall. 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, which unfortunately 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 user traffic.

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 than under the tail-based policy, which dominates the average parity update time and

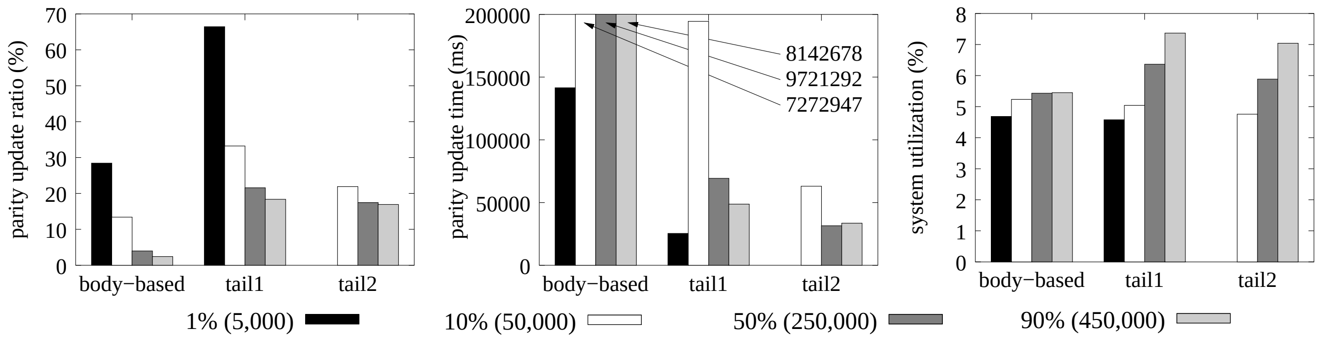

Figure 4. Performance of parity updates for trace T2 (high variability) and four different user WRITE traffic, i.e., 1%, 10%, 50% and 90% (numbers in parenthesis indicate the absolute number of user WRITEs).

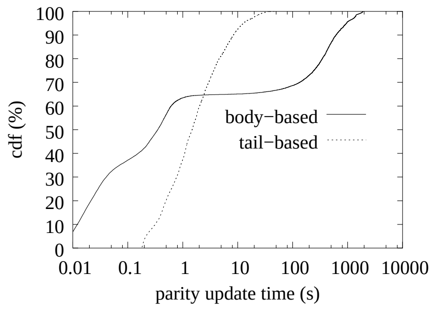

Figure 3. CDF of parity updates time for trace T1 (low variability).

Trace	Policy	User Issued WRITEs				
		1	2	3	4	5
T1	body	0.65	0.16	0.04	0.10	0.01
	tail	0.44	0.29	0.09	0.12	0.02

Table 7. Probabilities of user WRITES in trace T1 (low variability) that find dirty parity.

causes a two orders of magnitude advantage for the average tail-based performance.

Because parity updates are postponed in idle periods, some user WRITEs may find dirty parity in their corresponding parity segment. Updating parity when multiple WRITEs have occurred in the parity segment is more prone to errors than when only one WRITE has been completed. Table 7 gives the probabilities that by the time a parity is updated, the corresponding parity segment has been overwritten up to five times by the user. Although the metric depends on parity update times, it also depends on the spatial locality of the user WRITE workload. Trace T1 does have this characteristic. Results in the table show that the majority of parity updates (approximately 75%) for both policies occur when the segment has been written at most twice.

7.2 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.

Figure 4 presents parity update performance for trace T2 (and its variants) using the body-based and tail-based policies to schedule work in idle times. Figure 4 shows two different performances for the tail-based policy (marked in the plots as “tail1” and “tail2”). Although both tail-based policies utilize the tail of the idle times, under “tail1” the idle wait is (approximately 40%) shorter than under “tail2”.

Because T2 has highly variable idle times, the tail-based policy outperforms the body-based one. For example, the body-based policy performs at least two to three times worse than the tail-based policy with respect to the total number of completed parity updates and 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, “tail2” achieves better update time while “tail1” achieves better number of completed updates. Timely updates are critical for MTTDL, we elaborate more on this later in this section.

The overall system utilization in Figure 4 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 5 plots the CDFs of parity update times for all four variants of trace T2. Consistently with results in Fig-

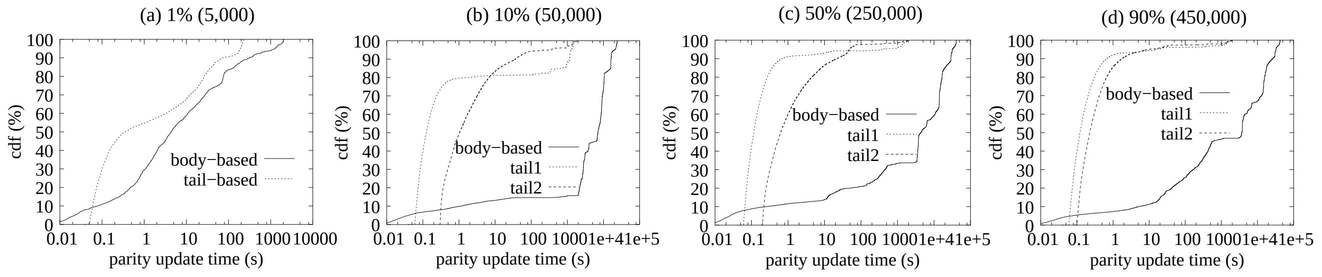

Figure 5. CDF of parity update time for trace T2 (high variability) and four different user WRITE traffic, i.e., 1%, 10%, 50% and 90% (numbers in parenthesis indicate the absolute number of user WRITEs).

Trace	Policy	User Issued WRITEs				
		1	2	3	4	5
T2 (1%)	body	0.98	0.02	N/A	N/A	N/A
	tail	0.99	0.01	N/A	N/A	N/A
T2 (10%)	body	0.75	0.17	0.05	0.01	0.01
	tail	0.85	0.12	0.02	0.006	0.001
T2 (50%)	body	0.53	0.22	0.10	0.05	0.03
	tail	0.65	0.22	0.07	0.03	0.01
T2 (90%)	body	0.46	0.21	0.11	0.06	0.04
	tail	0.59	0.24	0.08	0.03	0.02

Table 8. Probabilities of user WRITES in trace T2 (high variability) that find dirty parity.

ure 4, under the body-based policy the distribution has longer tail than under the tail-based policy. The "tail2" 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 "tail2" variant has also the longest idle waiting, which indicates that it uses the smallest number of idle intervals among all policies evaluated and has to wait for the very long intervals to arrive. Nevertheless, it results in the shortest average and distribution tail for update times. As parity updates increase in number, the differences in the distribution of update times between "tail1" and "tail2" decrease.

Table 8 presents the probabilities that by the time a parity update occurs, up to five user WRITEs have modified the parity segment for all four variants of trace T2. As the portion of user WRITEs increases in the trace, the probability of one user WRITE updates decreases. The body-based policy results are consistently worse than the results under the tail-based policies. In the best case (i.e., 1% WRITEs) 100% of parity updates happen when the parity segment has been modified at most twice for both policies.

7.3 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, the improvement in the mean interarrival time of latent sector errors is 0.48×10^2 [4], 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, and $ML^{(2)}$ represents the mean interarrival time of latent errors if there is intra-disk data redundancy.

If instantaneous parity (IP) is supported (i.e., parity updates occur without delay), then MTTDL is calculated using Equation (1) and $ML^{(2)}$ is used in place of ML, i.e.,

$$MTTDL = MTTDL_{ML^{(2)}}.$$

If parity updates are delayed, then Equation (1) is modified as follows:

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

where p represents the probability that the parity is dirty and $MTTDL_{ML^{(1)}}$ is computed using Equation (1) and value $ML^{(1)}$ is used for ML. 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:

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

where 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 processes during idle intervals determines RT_{Update} and consequently affects the MTTDL.

Policy	T1	T2			
		1%	10%	50%	90%
body	48.1	48.4	46.6	38.6	35.1
tail1	48.4	48.4	48.3	48.2	48.2
tail2	N/A	N/A	48.4	48.3	48.3
IP	48.4	48.4	48.4	48.4	48.4

Table 9. MTTDL improvement via intra-disk data redundancy.

Assuming that the disk capacity is 40GB, the relative MTTDL improvement is estimated for parity updates for trace T1 and the four variants of trace T2. Results are given in Table 9. The improvement attributed to intra-disk parity are only one order of magnitude – recall that those attributed to scrubbing are as high as four orders of magnitude. The important result of Table 9 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.

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 performance of these two features when running concurrently in idle times, dubbed as "scrubbing+parity". Because both features run in 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. This means that 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 Results for Trace T2

Initially, we present results for T2. As for this trace, both scrubbing and parity updates individually perform better using the tail-based policy, Figures 6(a) and (b) give the average scrubbing and parity update times under this policy. For comparison, in each plot the results of only disk scrubbing and only intra-disk parity are also included. 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% 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., 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.

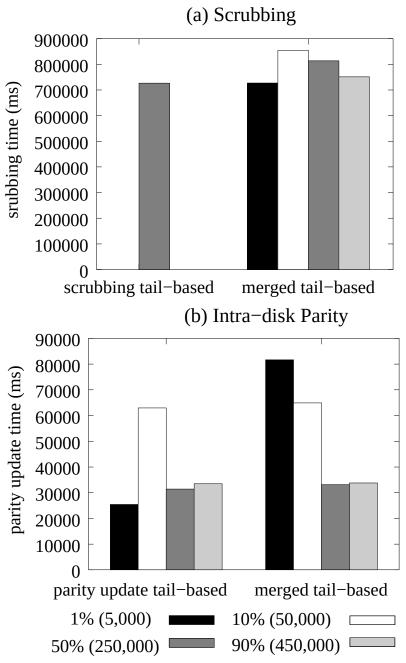

Figure 6. Average (a) scrubbing and (b) parity update times when running individually and together.

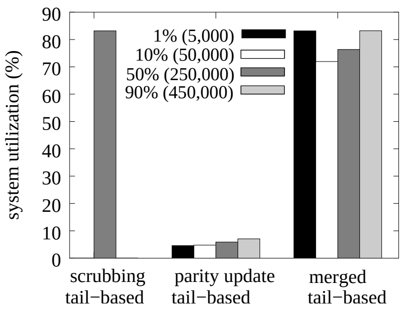

Figure 7. Overall system utilization under scrubbing and parity updates when they run individually and together.

Figure 7 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.

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 using the body-based policy while parity updates are done more efficiently using the tail-based policy. Thus, in addition to the body-based and the tail-based policies for the combined background work, we also evaluate another scheduling policy which schedules scrubbing work via the body-based policy and parity-updates via the tail-based policy. This policy is dubbed "body+tail-based" policy.

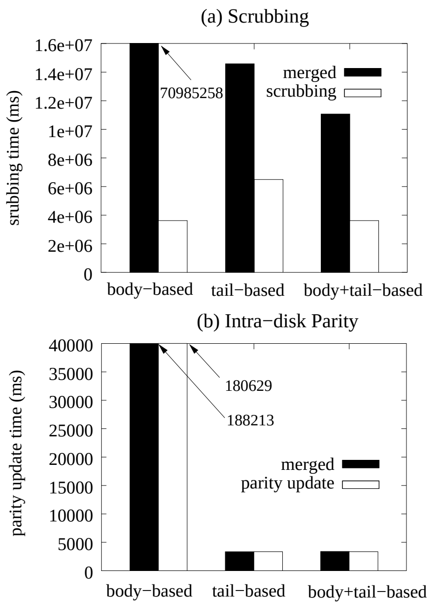

Figure 8. Average time for (a) an entire scrubbing, (b) parity updates for trace T1 (low variability). The body+tail policy schedules scrubbing via the body-based policy and parity-updates via the tail-based policy

Figure 8(a) presents the average time for a complete scrubbing when run individually (and 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-based variation, each background activity (i.e., scrubbing or parity update) is scheduled using the policy under which it performs best when running individually. Parity updates, because they have higher priority than scrubbing, are not penalized as much as scrubbing (see Figure 8(b)). Furthermore, parity updates perform significantly better if they are scheduled using the tail-based policy, independently of how scrubbing is scheduled.

Figure 9 presents system utilization for trace T1. Results are in agreement with those shown in Figure 8, the body+tail-based policy utilizes best the entire system providing room for both scrubbing and parity updates to perform similar to their best individual performance.

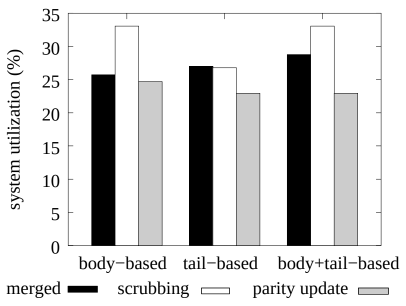

Figure 9. Overall system utilization

Policy	T1	T2			
		1%	10%	50%	90%
body	1.8 $\times 10^7$	N/A	N/A	N/A	N/A
tail	6.3 $\times 10^7$	1.12 $\times 10^8$	1.09 $\times 10^8$	1.07 $\times 10^8$	1.04 $\times 10^8$
body+ tail	7.1 $\times 10^7$	N/A	N/A	N/A	N/A

Table 10. MTTDL improvement via scrubbing and intra-disk parity.

8.3 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 $ML^{(2)} = 0.48 \times 10^2 \cdot ML^{(1)}$. Also assuming MRL ≈ 0. The parameter p in Equation (3) is estimated using Equation (4) and the average parity update time when it runs concurrently with scrubbing. Results are presented in Table 10. For trace T1 and four variants of trace T2, the MTTDL improvement attributed to scrubbing and intra-disk parity are as high as 7 and 8 orders of magnitude, respectively. Consistently with the results shown in Figure 8, the body+tail-based policy achieves better improvement in the MTTDL than both the body-based and the tail-based policies.

9 Conclusions

In this paper, we evaluate the effects that idle time scheduling policies have on the performance of background activities when the former is constrained by the degradation in user performance. The focus is on two data loss prevention techniques, i.e., disk scrubbing and intra-disk data redundancy. Scrubbing (representing infinite amount of background work) and parity-updates related to intra-disk redundancy (representing finite amount of background work) are evaluated here via trace-driven simulations.

Our evaluation shows that exploiting variability in the length of idle periods allows meeting user performance requirements and satisfactory background performance when measured by the data reliability enhancement it provides. This approach also allows to schedule both background activities simultaneously and still meet user performance targets. Each of the evaluated background features improves data reliability by orders of magnitude. The enhancement on data reliability when both background activities are scheduled in the system is higher than the linear combination of their individual benefits.

References

[1] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Lazy verification in fault-tolerant distributed storage systems. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 179–190. IEEE Press, October 2005.

[2] E. Bachmat and J. Schindler. Analysis of methods for scheduling low priority disk drive tasks. In *ACM Conference on Measurements and Modeling of Computer Systems (SIGMETRICS)*, pages 55–65. ACM Press, June 2002.

[3] 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.

[4] 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, Technical Report RZ3652, IBM Reasearch, 2006.

[5] 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.

[6] 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, Brighton, UK, October 2005.

[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 *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 263–276, New York, NY, USA, 2005. ACM Press.

[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] Michael J. Litzkow, Miron Livny, and Matt W. Mutka. Condor - a hunter of idle workstations. In *the 8th International Conference on Distributed Computing Systems (ICDCS)*, pages 104–111, 1988.

[13] 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.

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

[15] N. Mi, A. Riska, Q. Zhang, E. Smirni, and E. Riedel. Efficient utilization of idle times. Technical report, Technical Report 2006-10-1, Seagate Research, 2006.

[16] Z. Niu, T. Shu, and Y. Takahashi. A vacation queue with setup and close-down times and batch markovian arrival processes. *Perform. Evaluation*, 54(3):225–248, 2003.

[17] 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.

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

[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. IEEE Press, 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] H. Takagi. *Queuing Analysis Volume 1: Vacations and Priority Systems.* North-Holland, New York, 1991.

[22] E. Xu and A. S. Alfa. A vacation model for the non-saturated readers and writers system with a threshold policy. *Performance Evaluation*, 50(4):233–244, 2002.

[23] 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.