Idle Read After Write - IRAW

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

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

Abstract

Despite a low occurrence rate, silent data corruption represents a growing concern for storage systems designers. Throughout the storage hierarchy, from the file system down to the disk drives, various solutions exist to avoid, detect, and correct silent data corruption. Undetected errors during the completion of WRITEs may cause silent data corruption. A portion of the WRITE errors may be detected and corrected successfully by verifying the data written on the disk with the data in the disk cache. Write verification traditionally is scheduled immediately after a WRITE completion (Read After Write - RAW) which is unattractive, because it degrades user performance. To reduce the performance penalty associated with RAW, we propose to retain the written content in the disk cache and verify it once the disk drive becomes idle. Although attractive, this approach (called IRAW - Idle Read After Write) contends for resources, i.e., cache and idle time, with user traffic and other background activities. In this paper, we present a trace-driven evaluation of IRAW and show its feasibility. Our analysis indicates that idleness is present in disk drives and can be utilized for WRITE verification with minimal effect on user performance. IRAW benefits significantly if some amount of cache, i.e., 1 or 2 MB, is dedicated to retain the unverified WRITEs. If the cache is shared with the user requests then a cache retention policy that places both READs and WRITEs upon completion at the most recently used cache segment, yields best IRAW performance without effecting user READs cache hit ratio and overall user performance.

1 Introduction

Nowadays the majority of the available information is digitally stored and preserved. As a result, it becomes critically important that this vast amount of data is available and accurate anytime it is accessed. Storage systems that host digitally stored data strive to achieve data availability and consistency. Data availability is associated mostly with hardware failures and redundancy is the common approach to address it. Data consistency is associated with hardware, firmware, and software errors. Redundancy is not sufficient to protect the data from corruption and sophisticated techniques, including checksumming, versioning, and verification, need to be in place throughout the storage hierarchy to avoid, detect, and successfully correct errors that cause data inconsistencies [9].

Generally, faults that affect data availability [20, 14] occur more often than errors that cause data corruption [2]. Consequently, data availability [13, 10] has received wider attention on storage design than data consistency [9]. Recent evaluation of a large data set [2] shows that the probability of an enterprise-level disk experiencing data corruption is low, i.e., only 0.06%. Nevertheless, when considering the large amount of digitally stored data, the occurrence rate of data corruption becomes non-negligible. As a result, ensuring data consistency has gained wide interest among storage system designers [2, 9, 12].

Data corruption often occurs during the WRITE process somewhere in the IO path. Consequently, techniques that avoid, detect, and correct data corruption are commonly associated with the management of WRITEs. Examples include the log-structured and journaling file systems [19, 22], data checksumming and identification at the file system level (i.e., ZFS) or controller level [12, 15], as well as WRITE verification anywhere in the IO path.

Traditionally, Read After Write (RAW) ensures the correctness of a WRITE by verifying the written content via an additional READ immediately after the WRITE completes. RAW degrades user performance significantly because it doubles the service time of WRITEs. As a result, RAW is activated at the disk drive level only during special circumstances, such as high temperatures, that may cause more WRITE errors. In this paper, we propose an effective technique to conduct WRITE verification at the disk drive level. Specifically, we propose Idle Read After Write - IRAW, which retains the content of a *completed* and *acknowledged* user WRITE request in the disk cache and verifies the on-disk content with the cached content during idle times. Using idle times for WRITE verification reduces significantly the negative impact this process has on the user performance. We show the effectiveness of IRAW via extensive trace-driven simulations.

Unlike RAW, IRAW requires resources, i.e., cache space and idle time to operate efficiently at the disk level. Cache space is used to retain the unverified WRITEs until the idle time becomes available for their verification. Nevertheless, in-disk caches of 16MB and underutilized disks (as indicated by disk-level traces) enable the effective operation of a feature like IRAW.

IRAW benefits significantly if some amount (i.e., 2 MB) of dedicated cache is available for the retention of the unverified WRITEs. Our analysis shows that even if the cache space is fully shared between the user traffic and the unverified WRITEs, a cache retention policy that places both READs and WRITEs at the most-recently-used position in the cache segment list, yields satisfactory IRAW performance, without affecting READ cache hit ratio, and consequently, user performance. We conclude that IRAW is a feature that with a priority similar to "best-effort" enhances data consistency at the disk drive level, because it validates more than 90% of all the written content even in the busiest environments.

The rest of the paper is organized as follows. Section 2 discusses the causes of data corruption and focuses on data corruption detection and correction at the disk drive level. We describe the WRITE verification process in Section 3. Section 4 describes the disk-level traces used in our evaluation and relates their characteristics to the effectiveness of detection and correction of data corruption at the disk drive level. In Section 5, we present a comprehensive analysis of WRITE verification in idle time and its effectiveness under various resources management policies. Section 6 presents a summary of the existing work on data availability and reliability, in general, and data consistency, in particular. We conclude the paper with Section 7, which summarizes our work.

2 Background

In this section, we provide some background on data corruption and ways to address it at various levels of the IO path. Generally, data corruption is caused during the WRITE process because of various causes. Data corruption occurs when a WRITE, even if acknowledged as successful, is erroneous. WRITE errors may lead to data being stored incorrectly, partially, or not in the location where it is supposed to be [9]. These WRITE errors are known as lost WRITEs, torn WRITEs, and misdirected WRITEs, respectively. The cause of such errors may be found anywhere in the storage hierarchy.

Traditionally, data inconsistencies have been linked with the non-atomicity of the file system WRITEs [19, 22]. A file-system WRITE consists of several steps and if the system crashes or there is a power failure while these steps are being carried out, the data may be inconsistent upon restarting the system. Legacy file systems such as log-structured and journaling file systems address data inconsistencies caused by system crashes and power failures [19, 22].

However, data corruption may be caused during the WRITE process by errors (bugs) in the software or firmware throughout the IO path, from the file system to the disk drives, or by faulty hardware. Although erroneous, these WRITEs are acknowledged as successful to the user. These errors are detected only when the data is accessed again and as a result these errors cause *silent* data corruption. WRITE errors that cause silent data corruption are the focus of this paper. Addressing data inconsistencies because of power failures or system crashes are outside the scope of our paper.

Errors that cause silent data corruption represent a concern in storage system design, because, if left undetected, they may lead to data loss or even worse deliver inaccurate data to the user. Various checksumming techniques are used to detect and correct silent data corruption in the higher levels of the IO hierarchy. For example, ZFS [12] uses checksumming to ensure data integrity and consistency. Similarly, at the storage controller level checksumming techniques are coupled with the available data redundancy to further improve data integrity [9]. Logical background media scans detect parity inconsistencies by accessing the data in a disk array and building and checking the parity for each stripe of data [1].

Disk drives are responsible for a portion of WRITE errors that may cause silent data corruption in a storage system. WRITE errors at the disk drive may be caused by faulty firmware or hardware. The written content is incorrect although the completion of the WRITE command is acknowledged as successful to the user. Disk drives can detect and correct the majority of the disk-level WRITE errors via WRITE verification. In particular, disk drives can detect and correct WRITE errors when data is written incorrectly, partially or not at all at a specific location. WRITE verification at the disk level does not help with misdirected WRITEs, where the content is written somewhere else on the disk or on another disk in a RAID array.

3 Disk-level WRITE Verification

At the disk level, WRITE errors can be detected and recovered by verifying that the WRITE command was really successful, i.e., by comparing the written content with the original content in the disk drive cache. If inconsistency is found, then the data is re-written. WRITE verification can be conducted only if the written data is still in the disk cache. As a result, WRITE verification can occur immediately upon completion of a WRITE or soon thereafter. If the verification occurs immediately upon a WRITE completion, the process is known as WRITE Verify or Read-After-Write (RAW). RAW has been available for a long time as an *optional* feature in the majority of hard drives. Its major drawback is that it requires one additional READ for each WRITE, doubling the completion time of WRITEs (in average). Consequently, RAW is turned on only if the drive operates in extreme conditions (such as high temperature) when the probability of WRITE errors is high.

If the recently written data is retained in the disk cache even after a WRITE is completed, then the disk may be able to verify the written content at a more opportune time, such as the disk idle times (when no user requests are waiting for service). This technique is called Idle READ After WRITE (IRAW). Because disk arm seeking is a non-instantaneously preemptable process, the user requests will be delayed even if verifications happen in idle time, albeit the delay is much smaller than under RAW. As a result IRAW represents a more attractive option to WRITE verification at the disk drive level than RAW.

There is a significant difference between RAW and IRAW with regard to the resource requirements these two features have. RAW does not require additional resources to run, while IRAW is enabled only if there are resources, namely cache and idle time, available at the disk drive. The main enabler for IRAW in modern disk drives is the large amount of the available in-disk cache. The majority of disk drives today have 16 MB of cache space. The existence of such amount of cache enables the drive to retain the recently written data for longer, i.e., until the disk drive becomes idle, when the WRITE verification causes minimal performance degradation on user performance.

The effectiveness of IRAW depends on effective management of the available cache and idle time. Both cache and idle time represent resources that are used extensively at the disk drive, and IRAW will contend with other features and processes to make use of them both. For example, in-disk cache is mainly used to improve READ performance by exploiting the spatial and temporal locality of the workload, i.e., aggressively prefetching data from the disk or retaining recent READs in the cache hoping that incoming requests will find the data in the cache and avoid costly disk accesses. On the other hand, idle time is often used to deploy features that enhance drive operation such as background media scans. IRAW should not fully utilize the idle time and limit the execution of other background features.

On average, disk drives exhibit low to moderate utilization [17], which indicates that idle intervals will be available for WRITE verifications. Furthermore, in low and moderate utilization, busy periods are short as well. As a result only a few WRITEs will need to be retained in the cache, and wait for verification during the incoming idle period. Consequently, IRAW cache requirements are expected to be reasonable. However, the disk drive workloads are characterized by bursty periods [18] which cause temporal resource contention and inability to complete WRITE verifications. In this paper, we focus on the evaluation of IRAW and ways to manage resources, i.e., cache and idle time, such that IRAW runs effectively, i.e., the highest number of WRITEs is verified with minimal impact on user performance. Our focus is on four key issues:

- the available idle time for IRAW,
- the impact of IRAW on the performance of user requests, because they arrive during a non-preemptive WRITE verification,
- the cache requirements that would enable IRAW to verify more than 90% of all WRITEs in the workload,
- the impact that retention of unverified WRITEs in the cache has on READ cache hit ratio.

4 Trace Characterization

The traces that we use to drive our analysis are measured in various enterprise systems. These systems run dedicated servers that are identified by the name of the trace. Specifically, we use five traces in our evaluation; the “Web” trace measured in a web server, the “E-mail” trace measured in an e-mail server, the “Code Dev.” trace measured in a code development server, the “User Acc.” trace measured in a server that manages the home directory with the accounts of the users in the system, and the

Trace	Length (hrs)	Idle %	Avg. Idle Int. (ms)	R/W Ratio
Web	7	96	274	44/56
E-mail	25	92	119	99/1
User Acc.	12	98	625	87/13
Code Dev.	12	94	183	88/12
SAS	24	99	88	40/60

Table 1: General characteristics for disk-level traces used in our analysis.

"SAS" trace measured in a server running the SAS statistical package. Several of the measured storage subsystems consist of multiple disks, but throughout this paper, we focus on traces corresponding to the activity of single disks. Traces record several hours of disk-level activity (see Table 1) which make them representative for the purpose of this evaluation.

Traces record for each request the disk arrival time (in ms), disk departure time (in ms), request length (in bytes), request location (LBA), and request type (READ or WRITE). Here, we focus mostly on characteristics that are relevant to IRAW. General characterization of the traces as well as how they were collected can be found in [17, 18]. The only information we have on the architecture of the measured systems is the dedicated service they provide and the number of disks hosted by the storage subsystem.

Several trace characteristics such as arrival rate, READ/WRITE ratio, idle and busy time distributions are directly related to the ability of the disk drive to verify WRITEs during idle intervals. In Table 1, we give the general characteristics (i.e., trace length, disk idleness, average length of idle intervals, and READ/WRITE ratio) of the traces under evaluation. While READ/WRITE ratio is derived using only the information on the *request type* column of each trace, the idleness and idle interval lengths are calculated from the information available in the arrival time and departure time columns. The calculation of system idleness as well as the length of idle and busy periods from the traces is exact (not approximate), and facilitates accurate evaluation of IRAW.

Table 1 indicates that disk drives are mostly idle, which represents a good opportunity for IRAW to complete successfully during idle times. The average length of idle intervals indicates that several WRITEs may be verified during each idle interval. The READ/WRITE ratio in the incoming user traffic indicates the portion of the workload that needs verification in idle times and determines the IRAW load. Because the READ/WRITE ratio varies in the traces of Table 1, the IRAW performance will be evaluated under different load levels. Although the application is the main determining factor of the READ/WRITE ratio of disk-level workloads, the storage system architecture plays an important role as well. For the systems where the Web and the SAS traces were measured, the IO path has less resources and, consequently, intelligence than the other three traces. We came to this conclusion because the Web and SAS traces are measured on storage subsystems with single disks while the other traces are measured on storage subsystems with multiple disks. This leads us to believe that, except the Web and the SAS systems, the measured storage subsystems are organized in RAID arrays. Also from the traces, we can extract information on the WRITE optimization that takes place above the disk level. WRITE optimization consists of coalescing, usage of non-volatile caches, and other features, which reduce overall WRITE traffic. An indication, at the disk level, of the presence of non-volatile caches or other WRITE optimization features in the IO path, (see [17] for longer discussion), is the frequency of re-writes on a recently written location. While for the E-mail, User Acc. and Code Dev. traces the written locations are not re-written for the duration of each trace, for the Web and SAS traces this is not the case.

Figure 1 gives the arrival rate (i.e., the number of requests per second) as a function of time for several enterprise traces from Table 1. The disk-level workload is characterized by bursts in the arrival process. The arrival bursts are sometimes sustained for long (i.e., several minutes) periods of time. Arrival bursts represent periods of time when resources available for IRAW (i.e., cache and idle time) are limited. Consecutively, it is expected that IRAW will not have enough resources to verify all WRITEs in an environment with bursty workloads.

In Figure 2, we present the distribution of idle periods for the traces of Table 1. In the plot, the x-axis is in log-scale to emphasize the body of the distribution that indicates the common length of the idle intervals. Almost 40% of the idle intervals in the traces are longer than 100 ms and only one in every three idle intervals is less than a couple of milliseconds. Such idle time characteristics favor IRAW and indicate that in each idle interval, the drive will be able to verify at least several WRITEs.

The minimum length of the idle intervals, as well as their frequency is a useful indicator in deciding the *idle waiting* period, i.e., the period of time during which the drive remains idle although IRAW can be performed. Idle waiting is a common technique to avoid utilizing very short idle intervals with background features like IRAW and to minimize the effect disk-level background features have on user performance [4]. The case when a new user request arrives while a WRITE is being verified represents the case when IRAW degrades the performance of user requests. Figure 2 clearly indicates that

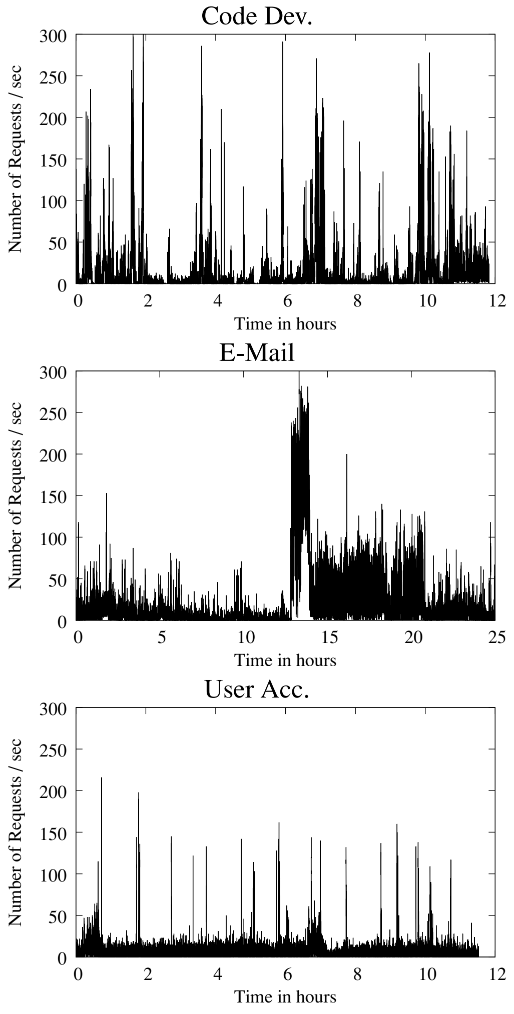

Figure 1: Arrival rate, measured in number of requests per second, as a function of time for several traces from Table 1. The arrivals are bursty in enterprise systems.

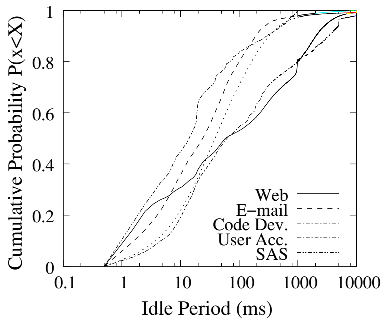

Figure 2: Distribution of idle periods for the traces of Table 1. X-axis is in log scale. The higher the line the shorter the idle periods are for the specific trace.

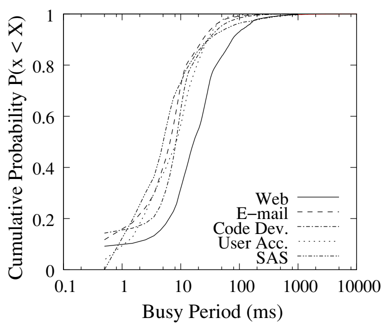

Figure 3: Distribution of busy periods for the traces of Table 1. The x-axis is in log-scale. The higher the line the shorter the busy periods are for the specific trace.

more than 90% of all idle intervals in all evaluated traces are longer than 10 ms, which leads us to optimistically state that by waiting a couple of milliseconds in an idle drive before a WRITE verification starts, the impact on the user requests performance will be contained to minimum.

IRAW effectiveness depends not only on the available idleness and length of idle periods, but also on the length of the busy periods at the disk drive level. The longer the busy period the larger the number of unverified WRITEs waiting for the next idle period and occupying cache space. In Figure 3, we present the distributions of busy periods for the traces of Table 1. Similarly to Figure 2, the x-axis of the plots is in log-scale. The distribution of the length of busy periods indicates that disk busy times are relatively short. Across all traces only 1% of busy periods are larger than 100 ms. The shape of the busy period distribution suggests that most WRITEs will get the chance to be verified during the idle period that immediately follows a busy period. Moreover, short busy intervals (Figure 3) and long idle intervals (Figure 2) indicate that IRAW will only use a fraction of the available idle time leaving room for additional background activities to be carried out too.

5 Evaluation of IRAW

The evaluation of IRAW is driven by the traces introduced in the previous section. Initially, we define a simplified version of IRAW, where (1) each WRITE verification takes the same amount of time, i.e., 5 ms, (2) there is dedicated cache available to store unverified WRITEs, and (3) the length of the idle interval is known, which means that WRITE verification will not affect the incoming user requests. With these assumptions, we can evaluate the effectiveness of IRAW directly from the traces and obtain an approximate estimation of the resource requirements for IRAW. We refer to this part of the evaluation as *trace-based* and discuss it in Section 5.1.

We further develop a simulation model for IRAW under the DiskSim 2.0 [6] disk-level simulator to relax the above assumptions and take into consideration the cache management aspect of IRAW. The simulation model is driven by the same set of traces. Because the simulation model represents an open model, we do not use the departure time field from the traces. As a result, the simulation model does not follow the idle and busy periods of the traces. The idle and busy periods in the simulation model are determined by the simulated disk, cache management policy, and the available cache size. We refer to this part of the evaluation as *simulation-based* and discuss it in Section 5.2.

In our evaluation, the efficiency of IRAW is measured by the *IRAW validation rate*, which represents the portion of WRITE requests verified during idle times. Any IRAW validation rate less than 100% indicates that not all WRITEs are verified. A WRITE is left unverified if it is evicted from the cache before idle time becomes available to verify it. Limited cache and/or limited idle time cause the IRAW validation rate to be less than 100%.

5.1 Trace-based Analysis

In the trace-based analysis, we assume full knowledge of the idle time duration, which means that IRAW will have no impact on the user performance for this type of analysis. We assume the validation of each WRITE takes the same amount of time to complete, i.e., 5 ms - the average time to complete a request at the disk drive. An unverified WRITE corresponds to the same WRITE request originally received by the drive, i.e., no coalescing or other techniques are used to reduce the number of unverified WRITEs. Verification is done in FCFS fashion.

Initially, we pose no restriction on the amount of available cache at the disk drive level. This assumption, although unrealistic, helps with the estimation of the maximum amount of cache required by IRAW to verify all WRITEs in the user workload. However, we do limit the amount of time an unverified WRITE waits in the cache for verification. We refer to this threshold as $IRAW_{Age}$ and measure it in number of idle intervals. An unverified WRITE waits through at most $IRAW_{Age}$ idle intervals before it is evicted from the cache. The threshold $IRAW_{Age}$ measures, indirectly, idle time availability at the disk drive level. That is, if a WRITE remains unverified through $IRAW_{Age}$ idle intervals, then, most probably, it will remained unverified in a more realistic scenario with limited cache space. The larger the $IRAW_{Age}$, the larger the maximum cache space requirements and the higher the IRAW validation rate.

Trace	IRAW Rate	IRAW$_{Age}$	Max Cache
Web	97 %	512	22 MB
E-mail	100 %	32	0.4 MB
User Acc.	100 %	64	1.7 MB
Code Dev.	100 %	256	8 MB
SAS	95 %	512	50 MB

Table 2: IRAW Verification Rate assuming unlimited cache and average verification time of 5 ms.

We set $IRAW_{Age}$ threshold to be 512, which means that the disk will retain an unverified WRITE through no more than 512 idle intervals. We measure the IRAW verification rate as a function of the $IRAW_{Age}$ and estimate the maximum amount of cache required to retain the unverified WRITEs until verification. We present our findings in Table 2.

Table 2 indicates that IRAW validation rate for 60% of the traces is 100%, with only moderate cache requirements, i.e., up to 8 MB of cache. For the traces that achieve 100% IRAW validation rate (i.e., E-mail, User Acc. and Code Dev.), the $IRAW_{Age}$ value is below the threshold of 512. This shows that, for these traces, there is idle time available to verify all WRITEs in the workload. From Table 1, we notice that the three traces that achieve 100% validation rate with moderate cache requirements have the lowest number of WRITEs in the workload. The other two traces, namely Web and SAS, have many more WRITEs in their workload mix. As a result, the verification rate is not 100%. Nevertheless, the Web and SAS traces achieve at least 95% IRAW validation rate. For these two traces, the amount of required cache space is high, i.e., more than 20 MB, which is unrealistic for a disk drive today. Following the discussion in Section 4 about the READ/WRITE ratio of traces in Table 1, recall that the high READ/WRITE ratio for Web and SAS may be associated with the IO path hierarchy in the systems where these traces were collected.

The results in Table 2, give a high level indication that the IRAW may be an effective feature, which will restrict performance degradation for user requests while maintaining high level of WRITE verification. However, because IRAW requires both cache and idle time to complete the verifications, the ratio of verified WRITEs, is not expected to be 100% in all cases.

The assumption of having unlimited cache is unrealistic. Hence, in the next experiment, we assume that the dedicated cache to IRAW is only 8 MB. By limiting the available cache size the $IRAW_{Age}$ threshold is eliminated, because now the reason for a WRITE to remain

Trace	Web	E-mail	User Acc.	Code Dev.	SAS
IRAW Rate	91%	100%	100%	100%	91%

Table 3: IRAW Verification Rate assuming 8 MB of available cache and average verification time of 5 ms.

unverified is the lack of cache to store it rather than the lack of idle time.

The corresponding results are presented in Table 3. As expected from the results in Table 2, IRAW verification rate for the E-mail, User Acc. and Code Dev. traces is still 100%. The other two traces, i.e., Web and SAS, perform slightly worse than in the case of unlimited cache (see Table 2). The Web and SAS traces require more than 20MB of cache space to achieve at least 95% IRAW verification rate. With only 8MB, i.e., almost three times less cache, IRAW validation rate is at least 91%. This result indicates that the maximum cache space requirement is related to bursty periods in the trace that reduce the availability of idle time for IRAW. Consequently, even in bursty environments where resources may be limited at time, there are opportunities to achieve high IRAW verification rates, i.e., above 90%.

5.2 Simulation-based Analysis

We use DiskSim 2.0 disk-level simulation environment [6] to evaluate in more detail the cache management strategies that work for IRAW. The simulation is driven by the same set of traces that are described in Section 4. The trace-based analysis provided an approximate estimation of IRAW cache space requirements, idleness requirements, as well as the overall IRAW validation rate. Section 5.1 concluded that in the enterprise environment, IRAW verifies at least 90% of WRITEs with moderate resource requirements (i.e., 8MB of cache) dedicated to IRAW.

The following simulation-based analysis intends to evaluate in more detail the cache management policies and how they effect IRAW performance and user request performance in presence of IRAW. The simulated environment is more realistic than the trace-based one, where several assumptions were in place. For example, in the simulation-based analysis, the idle interval length is not known beforehand and the verification time for WRITEs is not deterministic. Consequently, during the verification of a WRITE a user request may arrive and be delayed because the WRITE verification cannot be preempted instantaneously.

We simulate two disks, one with 15K RPM and 73GB of space and the second one with 10K RPM and 146GB of space, which model accurately the disks where the traces were measured. The latter disk is used to simulate only the Code Dev. trace from Table 1. Both disks are set to have an average seek time of 5.4 ms. The requests in both foreground and background queue are scheduled using the Shortest Positioning Time First (SPTF) algorithm. The IRAW simulation model is based on the existing components of the disk simulation model in DiskSim 2.0. The queue module in DiskSim 2.0 is used to manage and schedule the unverified WRITEs, and the cache module is used to manage the available cache segments between the user READs and WRITEs and the unverified WRITEs.

Trace	Max Cache	IRAW Rate	IRAW Response Time
Web	60 MB	100%	283 ms
E-mail	0.7 MB	100%	8 ms
User Acc.	2 MB	100%	10 ms
Code Dev.	60 MB	100%	5435 ms
SAS	48 MB	100%	1120 ms

Table 4: IRAW maximum cache requirements, verification rate, and verification response time, in our simulation model with unlimited cache space for unverified WRITEs.

As previously discussed, the trace-driven simulation results would reflect the modeling of scheduling, caching, and serving of user requests and will not fully comply with the results obtained from a trace-based evaluation only approach. Consequently, we do not expect exact agreement between the results in the trace-based evaluation of Subsection 5.1 and the simulation-based evaluation in this subsection.

Once the disk becomes idle, the WRITE verification process starts after 1 ms of idle time has elapsed. WRITE verifications are scheduled after some idle time has elapsed at the disk level to avoid utilizing the very short idle intervals and, consequently, limit the negative effect WRITE verification may have on user request performance. The benefit of idle waiting in scheduling low-priority requests such as WRITE verifications under IRAW are discussed in [4, 11]).

Initially, we estimate the maximum cache requirement for each of the traces under the simulation model. For this the simulation is run with no limitation on cache availability. The goal is to estimate how much cache is necessary to achieve 100% WRITE verification rate. Recall that the longer the unverified WRITEs are allowed to wait for validation the larger the required cache space to store them. The simulation results are presented in Table 4.

The results in Table 4 show that only for two traces (40% of all evaluated traces), IRAW achieves 100% validation rate by requiring a maximum of 2MB cache space. These two traces are characterized by low disk utilization (i.e., 99% idleness) or READ dominated workload (i.e., the E-mail trace has only 1.3% WRITEs). The other subset of traces (60% of them) requires more than 48MB of cache space, in the worst case, to achieve 100% IRAW verification rate. The worst WRITE verification response time in these traces is 5.4 sec, which explains the large cache requirements. The results of Table 4 are qualitatively the same as the one Table 2. IRAW verification rate of 100% comes with impractical cache requirements for half of the traces.

In an enterprise environment, IRAW is expected to require large cache space in order to achieve 100% IRAW validation rate, because the workload, as indicated in Section 4, is characterized by bursts. The bursts accumulate significant amount of unverified WRITEs in short periods of time. These WRITEs need to be stored until the burst passes and the idleness facilitates the verification.

Table 4 shows also the average IRAW response time, i.e., the time unverified WRITEs are retained in the cache. For the traces that capture light load, i.e., E-mail and User Acc. traces, the WRITEs are verified without waiting too long, similar to how RAW would perform. For the traces that capture medium to high load, i.e., Code Dev. and SAS traces, the IRAW response time is up to several seconds, which indicates that the unverified WRITEs will occupy the available cache for relatively long periods of time.

Although IRAW is designed to run in background, it will, unavoidably, impact at some level the performance of the user requests, i.e., foreground work. There are two ways that IRAW degrades foreground performance

- Upon arrival, a new request finds the disk busy verifying a WRITE when otherwise the disk would have been idle. Because the WRITE validation cannot be interrupted once started, the response time of the newly arrived user request and of any other user requests in the incoming foreground busy period will be longer by the amount of time between the first user requests arrival and the completion of WRITE verification. The WRITE verification as any other disk-level service is non-instantaneously preemptable because seeking in the disk drive is non-preemptable.

- Unverified WRITEs are stored in the disk cache to wait for an idle period when they can be verified. As a result, the unverified WRITEs occupy cache space, which otherwise would have been used by the user READ requests. As a consequence, IRAW may reduces READ performance by reducing the READ cache hit ratio.

Trace	Idle-ness	R/W Ratio	Max. diff	Avg. IOPS diff.
Web	96 %	44/56%	0.53%	0.02%
E-mail	92 %	99/1 %	0.11%	0.00%
User Acc.	98 %	87/13%	0.02%	0.00%
Code Dev.	94 %	88/12%	2.37%	0.08%
SAS	99 %	40/60%	0.12%	0.00%

Table 5: IRAW impact on system throughput measured by IOPS.

We analyze the impact of IRAW on the user performance by quantifying the reduction in the user throughput (measured by IOs per second - IOPS) and the additional wait experienced by the user requests because of the non-preemptability of WRITE verifications. We present our findings regarding the system throughput in Table 5 and the IRAW-caused delays in the user requests response time in Figure 4.

The trace-driven simulation model represents an open system. As a result the arrival times are fixed and will not change if the model simulates a disk slowed down by the presence of IRAW. This means that independent of the response time of requests, all requests will be served by the disk drive more or less within the same time period overall. This holds, particularly, because the traces represent cases with low and moderate utilization. As a result, to estimate the impact IRAW has on IOPS, we estimate the metric over short periods of time rather over the entire trace (long period of time) and focus on differences between the IOPS when IRAW is present and when IRAW is not present at the disk-level. We follow two approaches to estimate the IRAW caused degradation in IOPS. First we calculate the IOPS over 5 min intervals and report the worst case, i.e., the maximum IRAW-caused degradation in the IOPS over a 5 minutes interval. Second we calculate the IOPS for each second and report the average on the observed degradation. In both estimation methods, the impact that IRAW has on IOPS is low. We conclude that IRAW has minimal effect on system throughput for the evaluated traces from Table 5.

Results of Table 5 are confirmed by the distribution of the IRAW caused delays in the response time of user requests. The majority of user requests are not delayed by IRAW, as clearly indicated in Figure 4. For all traces, only less than 10% of user requests are delayed a few milliseconds, because they find the disk busy verifying

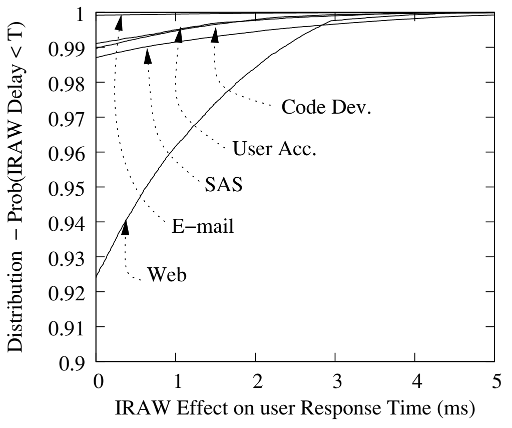

Figure 4: Distribution of IRAW caused delays.

WRITEs. For some traces such as the E-mail one, the delays are virtually non-existent. Since the average verification time is only a few milliseconds, the maximum IRAW-caused delays are also a couple of milliseconds as indicated by the x-axis of Figure 4.

In order to minimize the impact IRAW has on user performance, it is critical for IRAW to start WRITE verification only after some idle time has elapsed, called *idle wait*. In Figure 5, we show the IRAW validation rate for three different traces, as a function of cache size and length of the idle wait. The results suggest that an idle wait of up to 5 ms does not reduce the IRAW verification rate and does not affect the user requests performance. In our simulation model, we use the idle IRAW wait of 1 ms, but anything close to the average WRITE verification time of 3 ms yields similar performance.

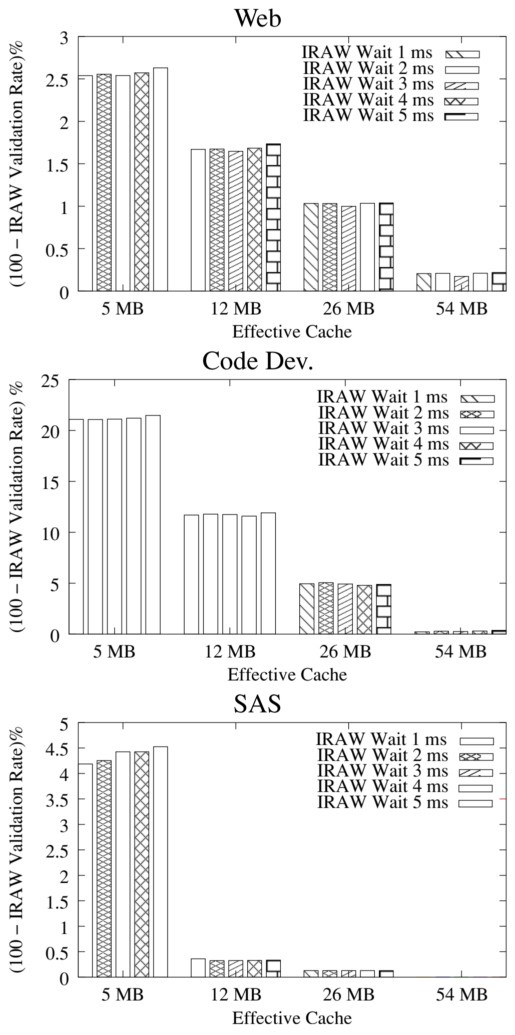

Figure 5: Impact of idle wait and cache space on IRAW performance.

5.3 Cache management policies

Disk drives today have approximately 16 MBytes of cache available. Disk caches are used to reduce the disk traffic by serving some of requests from the cache. The disk cache is volatile memory and because of data reliability concerns it is used to improve READ rather than WRITE performance by aggressive prefetching and data retention.

As a result, for background features like IRAW, which require some amount of cache for their operation, efficient management of the available cache is critical. While in the previous sections, we focused on evaluating IRAW and its maximum cache requirements, in this subsection, we evaluate IRAW performance under various cache management policies. We also estimate the impact that IRAW has on the READ cache hit ratio, which is directly related to the user performance.

There are two ways that IRAW uses the available cache space. First, IRAW shares the cache with the user READ traffic. In this case, READs and unverified WRITEs contend for the cache, with READs having more or at least the same priority as the unverified WRITEs. Second, IRAW uses dedicated cache space to store the unverified WRITEs. The IRAW dedicated cache space should enhance IRAW performance by minimally affecting READ cache hit ratio.

If IRAW and the READ user traffic share the cache, by default, IRAW has a “best-effort” priority, i.e., the lowest possible priority, because this is the priority of completed user WRITEs in the disk cache. This priority scheme gives no guarantees on IRAW verification rate. If some amount of dedicated cache space is allocated only for unverified WRITEs, then the IRAW priority is higher than just “best-effort”. Under this scheme, user READ requests will have less cache space available and, consequently, READ cache hit ratio will be lower. Overall, the IRAW validation rate is expected to be higher when dedicated cache space is allocated for unverified WRITEs than when IRAW contends for the available cache space with the READ user traffic.

The cache space in a disk drive is organized as a list of segments (schematically depicted in Figure 6). The head of the list of segments is the position from where the data is evicted from the cache. The head position is called the *Least Recently Used - LRU* segment and it has the lowest priority among all the cache segments. The further down a segment is from the LRU position, the higher its priority is and the further in the future its eviction time is. The tail of the segment list represents the segment with the highest priority and the furthest in the future eviction time. The tail position is referred to as the *Most Recently Used - MRU* position.

Commonly in disk drives, a READ is placed at the MRU position once the data is read from the disk to the cache, and a recently completed WRITE is placed at the LRU position. This policy indicates that for caching purposes, READs have the highest priority and WRITEs have the lowest priority. This is because a recently completed WRITE is not highly probable to be read in the near future. When a new READ occupies the MRU position, the previous holder of the MRU position is pushed up one position reducing its priority and the time it will be retained in the cache. All other segment holders are pushed up with one position as well, resulting in the eviction of the data from the LRU position. If there is a cache hit and a new READ request is accessing data found in the cache, the segment holding the data is placed in the MRU position and there is no eviction from the cache.

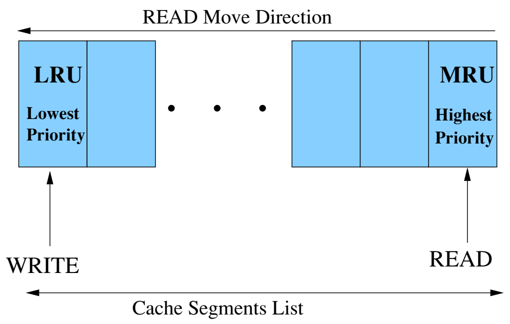

Figure 6: The model of the disk cache organized as a list of cache segments (each represented by a rectangle). The LRU position is the segment with the lowest retention priority and the MRU position is the segment with the highest retention priority. Upon completion, WRITES are placed at the LRU position and READS are placed at the MRU position.

The default cache retention policy does not favor the retention of unverified WRITEs. As a result, in the following, we investigate how the available cache may be shared between the READ user traffic and the unverified WRITEs such that both set of requests benefit from the available cache.

Initially, we evaluate the IRAW performance when it shares the cache space with the user READ traffic. We evaluate variations of the above default cache retention policy. A variation from the default cache retention policy is obtained by changing the default positions in cache for READs and unverified WRITEs upon the user request completion. The following cache retention schemes are evaluated:

- *the default*; READs are placed in the MRU position and unverified WRITEs in the LRU position (abbreviation: MRU/LRU),
- READs and unverified WRITES are both placed in the MRU position in a first-come-first-serve basis (abbreviation: MRU/MRU),
- READs and WRITEs are left in their current segments upon completion, i.e., a WRITE is not moved to the LRU position, a READ cache hit is not moved to the MRU position, a READ miss is placed in the MRU position (abbreviation: -/-).

Note that any cache retention algorithm other than those which place WRITEs in the LRU position upon completion, retain WRITEs longer in the cache and occupy space otherwise used by READs, which consecutively reduces the READ cache hit ratio, even though minimally. This is the reason why in our evaluation, the READ cache hit ratio and the IRAW validation rate are the metrics of interest. We analyze them as a function of the available data cache size.

In Figure 7, we present the cache hit ratio as a function of the cache size for several traces and cache retention policies. The plots of Figure 7 suggest that it is imperative for the READ cache hit ratio to place READs in the MRU position once the data is brought from the disk to the cache (observe the poor cache hit ratio for the "-/-" cache retention policy which does not change the position of a READ upon a cache hit). The fact that WRITEs are treated with higher priority by placing them into the MRU position too, leaves the READ cache hit ratio virtually unaffected. Another critical observation is that beyond some amount of available cache space, i.e., in all experiments approximately 12MB, the READ cache hit ratio does not increase indicating that adding extra cache space in a disk drive does not improve the READ cache hit ratio significantly, but can be used effectively for background features such as IRAW.

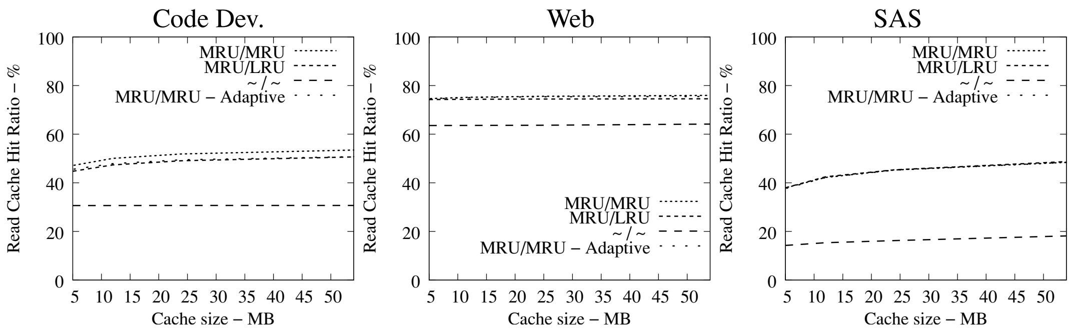

Figure 7: READ cache hit ratio as a function of cache size. Results are shown for various cache retention policies. A cache retention policy is identified by the placement of a READ (MRU, or no change) and the placement of unverified WRITEs (MRU, LRU, or no change).

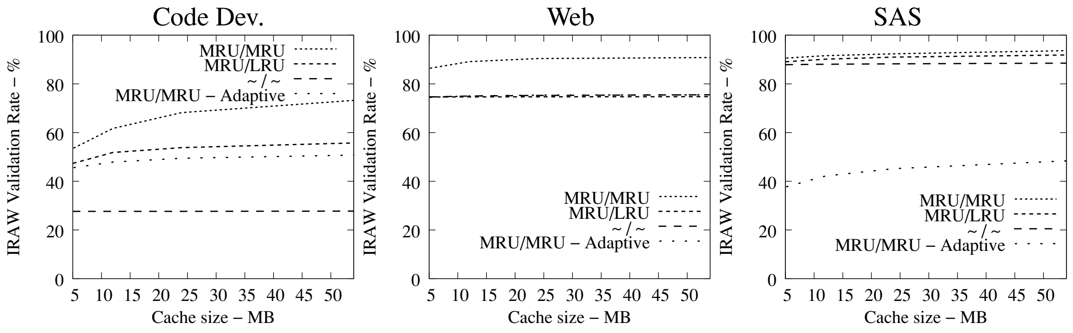

Figure 8: WRITE verification rate as a function of the cache size. Results are shown for various cache retention policies. A cache retention policy is identified by the placement of a READ (MRU, or no change) and the placement of unverified WRITEs (MRU, LRU, or no change).

In Figure 8, we present the IRAW validation rate as a function of the available cache, under various cache retention policies for several traces. IRAW is more sensitive to the cache retention policy than the READ cache hit ratio (see Figure 7). Placing unverified WRITEs in the MRU position is critical for the IRAW performance, in particular for the bursty case of the Code Dev. trace (recall that the simulated disk for the Code Dev. trace is a slower disk than for the rest of the enterprise traces). Figure 8 indicates that for most cases, i.e, 85% of them, shared cache retention algorithms work just fine and IRAW verification rate is above 90%.

In Figures 7 and 8, we also present results for an adaptive cache retention algorithm, where the READ/WRITE ratio of the workload is reflected on the amount of cache space used by READs and WRITEs. For example, a READ/WRITE ratio of 70%/30% would cause 70% of the cache space to be used by READs and 30% by the unverified WRITEs. As the ratio changes so does the usage of the cache. The adaptive policy improves the IRAW validation rate for most traces with almost no impact on READ cache hit ratio. However the gains are not substantial enough to justify the added complexity in the implementation of the adaptive cache retention algorithm.

Figure 7 suggests that READ cache hit ratio does not increase significantly as the available cache size increases beyond a certain point, i.e., in our analysis it is 10-12 MB. Consequently, we evaluate the effectiveness of IRAW when some amount of dedicated cache is allocated for the retention of the unverified WRITEs. In our evaluation, the user requests have the same amount of available cache for their use as well. For example, if IRAW will use 8MB of dedicated cache then so will the user READ requests. We present our results in Figure 9. Note that the plots in Figure 9 are the same as the respective ones in Figure 7 and Figure 8, but the "MRU/MRU - Adaptive" line is substituted with the "MRU/MRU - Dedicated" line. The results in Figure 7 indicate that the dedicated cache substantially improves the IRAW validation rate. This holds in particular for the heavy load cases such as the Code Dev. trace.

In conclusion, we emphasize that in order to maintain high READ cache hit ratio and high IRAW validation rate, it is critical for the available cache to be managed efficiently. Both READs and WRITEs need to be placed

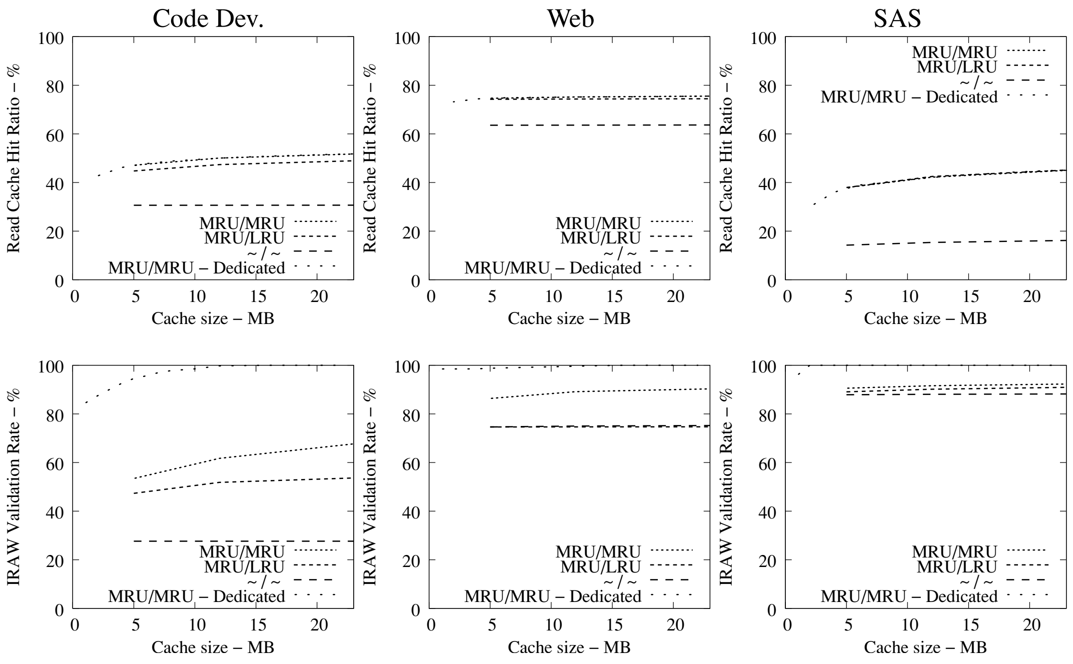

Figure 9: Read Cache hit ratio (first row) and IRAW validation rate (second row) as a function of the available dedicated cache.

in the MRU position upon completion. This cache retention policy yields the best performing IRAW for most environments, but not for the critical (very bursty) ones. The latter cases benefit enormously even if only a few MB of cache (i.e., 2 MB) are dedicated to store unverified WRITEs. Additional dedicated cache space for IRAW (i.e., 4-12MB) yields the best IRAW validation rate in the evaluated environments.

6 Related Work

Although disk drive quality improves from one generation to the next, they represent complex devices that are susceptible to a variety of failures [23, 24]. Because drive failures may lead to data loss, storage systems have widely accepted the RAID architecture [13, 10], which protects the data from one or two simultaneous failures. In theory storage systems can be designed to protect from n simultaneous disk drive failures, if $m > n$ disks are available [16]. Contemporary storage systems have adopted a distributed architecture with multiple copies of any piece of data [7] for added reliability, while using inexpensive disk drives.

As the amount of digitally stored data increases, so does the significance of storage and drive failures [20, 14]. In particular, rare failure events have become more prevalent. For example, in recent years significant effort has been devoted to better understand the effect of latent sector errors on overall data availability in storage systems [3, 5, 1]. Latent sector errors may happen at any time in a disk drive, but they may cause data loss (even of only a few sectors) if they remain undetected until another failure in the system (now with reduced data redundancy) triggers the entire data set to be accessed for reconstruction. To address such undesirable events, features like background media scans are added in storage system and disk drives [21, 1].

Traditionally, it has been the file system's task to ensure data consistency and integrity, assuming that the causes were related to power failure or system crashes during non-atomic WRITE operations. Legacy file systems address data consistency by implementing features like journaling and soft updates [22, 19]. Contemporary file systems [12, 15, 8] deploy more complex and aggressive features that involve forms of checksumming, versioning, identification for any piece of data stored in the system.

Today, storage system designers are concerned by silent data corruption. The growing complexity of systems enabling software, firmware, and hardware may cause data corruption and affect overall data integrity. Similar to disk latent sector errors, data corruption may happen at any time, but it can be detected only later on when the data is accessed. Such events may cause data loss, or, even worse, may deliver incorrect data to the user. Silent data corruption may occur in any component of the IO path.

Recent results from a large field population of storage systems [2] indicate that the probability that a disk develops silent data corruption is low, i.e., only 0.06% for enterprise-level disks and 0.8% for near-line disks. This occurrence rate is one order of magnitude less than the rate of a disk developing latent sector errors. Detection of silent data corruption as well as the identification of its source is not trivial and various aggressive features are put in place throughout the IO path to protect against silent data corruption [9].

Silent data corruption is associated with WRITEs and occurs when a WRITE, although acknowledged as successful, is not written in the media at all (i.e., lost WRITE), is written only partially (i.e., torn WRITE), or written in another location (i.e. misdirected WRITEs). The disk drive may cause some of the above WRITE errors. Read-After-Write (RAW) detects and corrects some WRITE errors by verifying the written content with the cached content. RAW may be deployed at the disk drive level or array controller level. RAW degrades significantly user performance and this paper focuses on effective ways to conduct WRITE verification.

7 Conclusions

In this paper, we proposed Idle Read After Write (IRAW), which verifies WRITEs at the disk drive level during idle time. IRAW aims at detecting and correcting any inconsistencies during the WRITE process that may cause silent data corruption and eventually data loss. Traditionally WRITE verification is conducted immediately after a WRITE completes via a process known as Read After Write. RAW verifies the content on the disk with the WRITE request in the disk cache. Because a WRITE is followed by an additional READ, RAW significantly degrades user's performance. IRAW addresses RAW's drawbacks by conducting the additional READs associated with a WRITE verification during idle time and minimizing the effect that WRITE verification has on user performance.

Unlike RAW, IRAW requires resources (i.e., cache and idle time) for its operation. Cache is required to store unverified WRITEs until idle time becomes available to perform the WRITE verifications. Nevertheless, in-disk caches of 16MB and underutilized disks (as indicated by disk-level traces) enable the effective operation of a feature like IRAW. Although IRAW utilizes only idle times, it effects user request performance, because it contends for cache with the user traffic and it delays user requests if they arrive during the non-preemptable WRITE verification. Consequently, we measure the IRAW performance by the ratio of verified WRITEs and the effect it has on user request performance.

We used several disk-level traces to evaluate IRAW's feasibility. The traces confirm the availability of idleness at the disk-level and indicate that disk's operation is characterized by short busy periods and long idle periods, which favor IRAW. Via trace-driven simulations, we concluded that IRAW has minimal impact on the disk throughput. The maximal impact on disk throughput measured over 5 minutes intervals is less than 1% for the majority of the traces. The worst estimated disk throughput degradation among the evaluated traces is only 2%.

Our evaluation showed that the cache hit ratio for the user traffic (and consequently user performance) is maintained if both READs and WRITEs are placed at the MRU (Most Recently Used) position in the cache upon completion. Because the READ cache hit ratio plateaus as the cache size increases, it is possible to use some dedicated cache space for IRAW without effecting READ cache hit ratio and improving considerably IRAW verification rate. Dedicated cache of 2MB seems to be sufficient to achieve as high as 100% IRAW validation rate for the majority of the evaluated traces. We conclude that IRAW is a feature that with a priority similar to "best-effort" enhances data integrity at the disk drive level, because it validates more than 90% of all the written content even in the burstiest environments.

References

[1] L. N. Bairavasundaram, G. R. Goodson, S. Pasupathy, and J. Schindler. An analysis of latent sector errors in disk drives. In *Proceeding of the ACM SIGMETRICS*, pages 289–300, 2007.

[2] L. N. Bairavasundaram, G. R. Goodson, B. Schroeder, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. An analysis of data corruption in the storage stack. In *to appear in Proceeding of the USENIX Annual Conference in File and Storage Systems*, 2008.

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

[4] L. Eggert and J. D. Touch. Idletime scheduling with preemption intervals. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP'05)*, pages 249–262, Brighton, UK, Oct. 2005. ACM Press.

[5] J. G. Elerath and M. Pecht. Enhanced reliability modeling of raid storage systems. In *DSN*, pages 175–184, 2007.

[6] G. R. Ganger, B. L. Worthington, and Y. N. Patt. The DiskSim simulation environment, Version 2.0, Reference manual. Technical report, Electrical and Computer Engineering Department, Cannegie Mellon University, 1999.

[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] H. S. Gunawi, V. Prabhakaran, S. Krishnan, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Improving File System Reliability with I/O Shepherding. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07)*, pages 283–296, Stevenson, Washington, October 2007.

[9] A. Krioukov, L. N. Bairavasundaram, G. Goodson, K. Srinivasan, R. Thelen, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Parity lost and parity regained. In *FAST*, 2008.

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

[11] N. Mi, A. Riska, Q. Zhang, E. Smirni, and E. Riedel. Efficient utilization of idle times. In *Proceedings of the ACM SIGMETRICS*, pages 371–372, 2007.

[12] S. Mirosystems. Zfs: the last word in file systems. Technical report, http://www.sun.com/2004-0914/feature, 2004.

[13] 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.

[14] E. Pinheiro, W.-D. Weber, and L. A. Barroso. Failure trends in a large disk drive population. In *FAST*, pages 17–28, 2007.

[15] V. Prabhakaran, L. N. Bairavasundaram, N. Agrawal, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. IRON File Systems. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, pages 206–220, Brighton, United Kingdom, October 2005.

[16] M. . Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of ACM*, 36(2):335–348, 1989.

[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] A. Riska and E. Riedel. Long-range dependence at the disk drive level. In *Proceedings of the Internatinal Conference on the Quantitative Evaluation of Systems (QEST)*, pages 41–50, 2006.

[19] M. Rosenblum and J. Ousterhout. The design and implementation of a log-structured file system. *ACM Transaction on Computer Systems*, 10(1):26–52, 1992.

[20] B. Schroeder and G. A. Gibson. Understanding disk failure rates: What does an mttf of 1,000,000 hours mean to you? *ACM Transactions on Storage*, 3(3), 2007.

[21] 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)*. IEEE Press, 2004.

[22] M. I. Seltzer, G. R. Ganger, M. K. McKusick, K. A. Smith, C. .A.Soules, and C. . Stein. Journaling versus soft updates: Asynchronous meta-data protection in file systems. In *Procceding of the 2000 USENIX Annual Technical Conference*, 2000.

[23] 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.

[24] J. Yang and F. Sun. A comprehensive review of hard-disk drive reliability. In *Procceding of the IEEE Annual Reliability and Maintainability Symposium*, pages 403–409, 1999.