An Adaptive High-Low Water Mark Destage Algorithm for Cached RAID5

Young Jin Nam and Chanik Park
Department of Computer Science and Engineering/PIRL
Pohang University of Science and Technology
Kyungbuk, Republic of Korea
{yjnam,cipark}@postech.ac.kr

Abstract

The High-Low Water Mark destage (HLWM) algorithm is widely used to enable a cached RAID5 to flush dirty data from its write cache to disks. It activates and deactivates a destaging process based on two time-invariant thresholds which are determined by cache occupancy levels. However, the opportunity exists to improve I/O throughput by adaptively changing the thresholds. This paper proposes an adaptive HLWM algorithm which dynamically changes its thresholds according to a varying I/O workload. Two thresholds are defined as the multiplication of changing rates of the cache occupancy level and the time required to fill and empty the cache. Performance evaluations with a cached RAID5 simulator reveal that the proposed algorithm outperforms the HLWM algorithm in terms of read response time, write cache hit ratio, and disk utilization.

1. Introduction

Despite its prevalence, RAID5 [4] has suffered from a *small write problem*, which refers to the phenomenon that RAID5 requires four disk accesses to write a single disk block: old parity/data reads and new parity/data writes. To overcome the small write problem in RAID5, several solutions have been proposed [1, 2, 5, 6, 7]. One utilizes the non-volatile write cache [1]. Since the service of a write request entails writing data onto the write cache, it can be performed quickly. The written (*i.e.*, dirty) data in the cache should be flushed into physical disks when the cache occupancy level arrives at a pre-defined threshold. The operation of flushing dirty data is called a destaging process, which is usually initiated by a destage scheduler or a destage algorithm.

The desirable properties that a destage algorithm encompasses are as follows. First, it has no effect on the response time of read requests. This implies that the destaging process is initiated when read requests are no longer in service. Second, it keeps the cache occupancy level as high as possible to increase the write cache hit ratio. In addition, keeping the cache occupancy level high may reduce the number of disk I/Os required for destaging. Third, it should avoid cache overflow, *i.e.*, maintain a sufficient number of clean or free cache entries. Observe that a trade-off should be made to pursue the last two properties at the same time. In addition, the following two issues ought to be taken into consideration to meet the properties noted above: how to elaborately determine its thresholds to manage the destaging process (e.g., starting or stopping destaging), and how to decide which dirty cache entries are to be destaged. This paper mainly emphasizes the first issue: how to efficiently determine the destaging thresholds.

Two destage algorithms have been available in terms of manipulating destaging thresholds: the High-Low Water Mark (HLWM) algorithm [1] and the Linear Threshold (LT) algorithm [7]. The HLWM algorithm is based on two destaging thresholds: the High Water Mark (HWM) and the Low Water Mark (LWM). It starts the destage process when the current cache occupancy level goes up to HWM. Conversely, it stops destaging when the cache occupancy level goes down to LWM. The LT algorithm defines more than two thresholds. As the current cache occupancy level reaches each higher threshold, it accelerates the destaging process by including more dirty entries in a single destaging write operation. Even though this algorithm outperforms the HLWM algorithm, it requires a complicated operation to compute the destaging costs for all the dirty entries at every disk head move. Observe that the HLWM and LT algorithms define their destaging thresholds as time-invariant cache occupancy levels. Next, the thresholds remain unchanged in the presence of a time-varying I/O workload. However, it is desirable to set HWM and LWM to higher levels under a light I/O workload in order to increase the write cache hit ratio. Conversely, it is required that HWM and LWM be set to lower levels under a heavy I/O workload, so as to prepare a sufficient number of free or clean cache entries. In sum, we claim that the opportunity exists to improve the I/O throughput by adaptively changing HWM and LWM according to a varying I/O workload.

In this paper, we propose an Adaptive High-Low Water Mark destage (AHLWM) algorithm which dynamically determines its thresholds according to a varying I/O workload. The thresholds are defined as the multiplication of changing rates of the cache occupancy level and the time required to fill and empty the cache. We prove that the AHLWM algorithm outperforms the HLWM algorithm in terms of read response time, write cache hit ratio, and disk utilization via performance evaluations on our cached RAID5 simulator. This paper is organized as follows: Section 2 describes the AHLWM algorithm and Section 3 evaluates its performance by comparison with the HLWM algorithm and then discusses an issue related to a bursty I/O workload. Finally, we conclude this paper with Section 4.

2. Adaptive High-Low Water Mark Destage Algorithm

This section describes the Adaptive High-Low Water Mark destage algorithm and then gives illustrative examples to explain its operation under various workload environments and configuration parameters.

Algorithm Description: Figure 1 compares a High-Low Water Mark destage (briefly HLWM) algorithm and an Adaptive High-Low Water Mark destage (AHLWM) algorithm in terms of managing their thresholds. Typically,

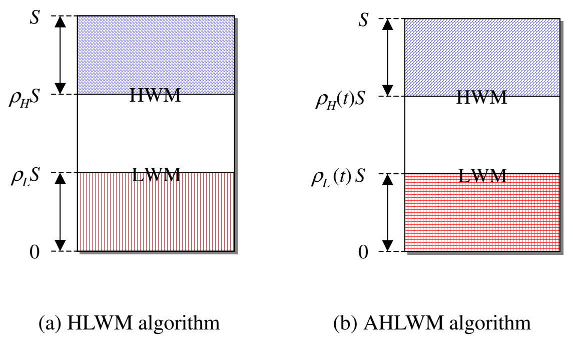


Figure 1. Comparisons of two thresholds of the HLWM algorithm and the AHLWM algorithm

the HLWM algorithm defines two static threshold values – High Water Mark (HWM) and Low Water Mark (LWM) by using cache occupancy levels, as shown in Figure 1(a). The HWM ($\mathcal{H}_{static}$) and LWM ($\mathcal{L}_{static}$) are written as

$$\mathcal{H}_{static} = \rho_H S \quad (1)$$
$$\mathcal{L}_{static} = \rho_L S, \quad (2)$$

where S is the cache size and $0 \leq \rho_L < \rho_H \leq 1$. Observe that both HWM and LWM of the HLWM algorithm remain unchanged during its operation under a time-variant I/O workload.

While the HWM and LWM are time-invariant in the HLWM algorithm, they are time-variant in the AHLWM algorithm, as shown in Figure 1(b). The HLM and LWM of the AHLWM algorithm can be written as follows:

$$\mathcal{H}_{adaptive}(t) = \rho_H(t)S, \quad (3)$$
$$\mathcal{L}_{adaptive}(t) = \rho_L(t)S. \quad (4)$$

Next, we will drive the time-variant $\rho_H(t)$ and $\rho_L(t)$. Let λ_B and μ_B represent base increasing and decreasing rates of the cache occupancy level. Note that λ_B is related to the increasing rate of the cache occupancy level, which accounts for the write cache hit and is not directly proportional to the arrival rate of write I/O requests. Let the current increasing and decreasing rates of the cache occupancy level at time t be $\lambda(t)$ and $\mu(t)$, In addition, we define an effective decreasing rate of the cache occupancy level at time t, $\alpha(t) = \mu(t) - \lambda(t)$. Similarly, $\alpha_B = \mu_B - \lambda_B$. Then, let us define T_H and T_L which represent the times required to fill and empty the cache as follows:

$$T_H = \frac{(1-\rho_H)}{\lambda_B} S, \quad (5)$$
$$T_L = \frac{\rho_L}{\lambda_B} S. \quad (6)$$

Given $\lambda(t)$ and $\alpha(t)$, the following equations should be valid:

$$\lambda(t)T_H + \rho_H(t)S = S, \quad (7)$$
$$\rho_L(t)S - T_L\alpha(t) = 0. \quad (8)$$

Finally, the time-variant $\rho_H(t)$ and $\rho_L(t)$ are written as:

$$\rho_H(t) = max\{0, 1-(1-\rho_H)\frac{\lambda(t)}{\lambda_B}\}, \quad (9)$$
$$\rho_L(t) = min\{1, max\{0, \rho_L\frac{\alpha(t)}{\alpha_B}\}\}. \quad (10)$$

Note that $\rho_H(t)$ can be smaller than $\rho_L(t)$ under a rare condition. To avoid this, the following condition has to be added in its implementation: $\rho_L(t) = \rho_H(t)$ if $\rho_H(t) < \rho_L(t)$.

Next, we will look into the useful properties of the AHLWM algorithm associated with the current increasing and decreasing rates. To begin, we can easily see that the following lemma holds without proof.

Lemma 1 *Given $\lambda(t) = \lambda_B$ and $\mu(t) = \mu_B$ at any time t, the AHLWM algorithm degenerates to the HLWM algorithm.*

Note that the two thresholds of the AHLWM algorithm are dynamically determined as the $\lambda(t)$ and $\mu_(t)$ change. However, the following inequality should be met at any given time t for normal operations: $\mathcal{H}_{adaptive}(t) \geq \mathcal{L}_{adaptive}(t)$.

Theorem 1 *$\mathcal{H}_{adaptive}(t) \geq \mathcal{L}_{adaptive}(t)$ at any time t if*

$$(1 - \rho_H)k \leq \lambda_B \leq \mu_B - \rho_L k, \tag{11}$$

for a given $\mu(t) = k$, where $0 \leq k \leq \frac{\mu_B}{1-(\rho_H - \rho_L)}$.

Proof: Basically, Equation (11) can be derived from two boundary conditions in Equation (9) and (10). It is easily seen that $\mathcal{H}_{adaptive}(t) \geq \mathcal{L}_{adaptive}(t)$ when $\lambda(t) = \lambda_B$ and $\mu(t) = \mu_B$ according to Lemma 1. Next, we will consider the two boundary conditions: 1) $\rho_L(t)|_{\lambda(t)=0} \leq \rho_H(t)|_{\lambda(t)=0} = 1$ and 2) $\rho_H(t)|_{\lambda(t)=k} \geq \rho_L(t)|_{\lambda(t)=k} = 0$. Then, we can derive the relation of $\lambda_B \leq \mu_B - \rho_L k$ from the boundary condition 1. Also, the $(1 - \rho_H)k \leq \lambda_B$ relation can be obtained from the boundary condition 2. Since $(1 - \rho_H)k \leq \mu_B - \rho_L k$, then we have $k \leq \frac{\mu_B}{1-(\rho_H - \rho_L)}$. It implies that the maximum value of k, *i.e.*, $\mu(t)$ is confined to $\frac{\mu_B}{1-(\rho_H - \rho_L)}$, called μ_B^{max}. Also, we can show that the two boundary conditions hold when $k = \mu_B^{max}$. Thus, this theorem follows. ▮

Corollary 1 *$\mathcal{H}_{adaptive}(t) \geq \mathcal{L}_{adaptive}(t)$ holds at any time t is satisfied if $\lambda_B = \frac{1-\rho_H}{1-\rho_H+\rho_L}\mu_B$.*

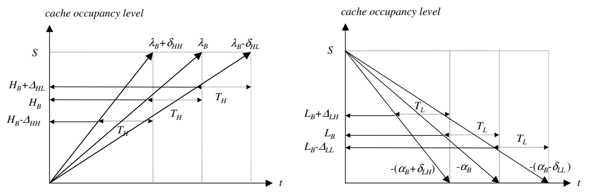
Proof: If $\mu(t) = \mu_B^{max}$ in Equation (11) of Theorem 1, the upper and lower bounds of λ_B converge into the point of $\lambda_B = \frac{1-\rho_H}{1-\rho_H+\rho_L}\mu_B$ which always makes the condition hold. Thus, this corollary follows. ▮

Corollary 2 *Given $0 \leq \mu(t) \leq \mu_B^{max}$, $\mathcal{H}_{adaptive}(t) \geq \mathcal{L}_{adaptive}(t)$ holds at any time t if*

$$(1 - \rho_H)\mu_B^{max} \leq \lambda_B \leq (1 - \rho_L)\mu_B^{max}. \tag{12}$$

Proof: The Equation (12) can be obtained by simply replacing k with μ_B^{max} in Equation (11) of Theorem 1. Thus, this corollary follows. ▮

Illustrative Examples: Figure 2 demonstrates the expected adaptiveness of HWM and LWM by the AHLWM algorithm according to the variation of an I/O workload. In Figure 2(a), $\mathcal{H}_B$ becomes $\mathcal{H}_B - \Delta_{HH}$ with a higher $\lambda(t) = \lambda_B + \delta_{HH}$ mainly attributed to a higher increasing rate of dirty data in the write cache. Conversely, $\mathcal{H}_B$ goes up to $\mathcal{H}_B + \Delta_{HL}$ with a lower increasing rate of $\lambda(t)$ by δ_{HL}. Similarly, $\mathcal{L}_B$ becomes $\mathcal{L}_B + \Delta_{LH}$ with a higher $\alpha(t) = \alpha_B + \delta_{LH}$ attributed to a higher rate of destaging data to disks or a lower increasing rate of dirty data in the write cache, as shown in Figure 2(b). Conversely, $\mathcal{L}_B$ goes down to $\mathcal{L}_B - \Delta_{LL}$ with the decrease of $\alpha(t)$ by δ_{LL}. Observe that Equation (5)–(8) remain valid under any I/O workload level.



(a) Variation of HWM (b) Variation of LWM

Figure 2. Variations of HWM and LWM according to different levels of I/O workloads, where S represents the cache size

3. Performance Evaluations under Cached RAID5

For performance evaluations of the AHLWM algorithm, we have implemented it within a cached RAID5 simulator which employs a write cache. Then, we compare the performance of the proposed algorithm with the HLWM algorithm. We begin by describing the cached RAID5 simulator.

3.1. Simulation Environments

The cached RAID5 simulator consists of a host I/O request queue, a host I/O scheduler, RAID I/O processes, a volatile read cache, a non-volatile write cache, disk I/O request queues, disk I/O schedulers, a destage process which includes different types of destage algorithms, and a synthetic I/O workload generator, as shown in Figure 3.

Overall I/O Operations: Host I/O requests issued by a group of workload generators is first be placed into the

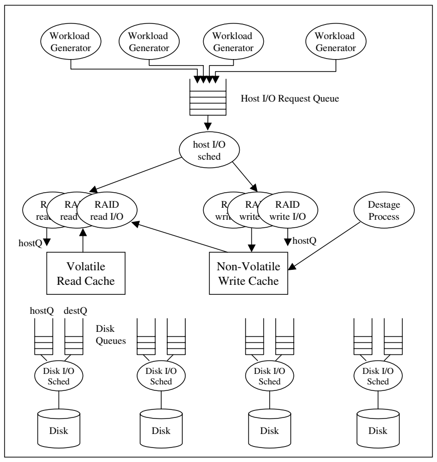


Figure 3. The internal structure of the cached RAID5 simulator

host I/O request queue in an FIFO manner. Next, the host I/O scheduler receives a host I/O request from the host I/O request queue. Then, it spawns either a RAID read I/O process or a RAID write I/O process, according to the request type. The RAID read I/O process searches its data first from the write cache and then from the read cache. If valid data resides in either of the two caches (a read cache hit), the cached data is immediately transferred to hosts with no disk I/O. If not (a read cache miss), the data is fetched from physical disks into the read cache and is then returned to hosts. The RAID write I/O process writes its data simply into the write cache and notifies its host of the completion of the current write request service. Later, the destage process flushes the newly written data called dirty data (entries) into physical disks. The behavior of the destage process will be described later in detail.

Each individual disk has a pair of queues which are the host request queue and the destage request queue. While disk I/O requests made from a read request are placed into the host request queue of a disk, the destage read/write I/O requests issued by the destage process are lined up in the destage queue of a disk. Note that a host request queue is given a higher priority in service than a destage queue, *i.e.*, the requests in the destage request queue are processed only while the host request queue becomes empty. In the event of a write cache overflow, the destage requests are placed into the host request queue. Each disk also contains its own I/O scheduler. Even though a few efficient disk scheduling algorithms are currently available, we employ a simple FIFO disk scheduling algorithm because performances of disk scheduling are not our concern.

Write I/O Operations and Destaging Process: To begin, Figure 4 depicts a state diagram of a cache entry within the write cache. Initially, the state of each entry is `NOT_USED` and `INVALID`. When the entry is allocated for storing dirty data from the host, its state is changed to `IN_USE` and `VALID/DIRTY`. If new dirty data arrives to the same location before the destaging of the dirty data, it is overwritten to the entry. This is called a write cache hit. Conversely, if new dirty data arrives during the destaging of the dirty data, it is stored separately by allocating different cache entries. At this point, its state is changed to `IN_USE` and `VALID/DDIRTY`. Subsequent dirty data at the same location will be overwritten, as previously. Once the destaging of a dirty data/entry completes, its state is changed to `NOT_USED` and `VALID/CLEAN`. In addition, if an associate cache entry of the `IN_USE` and `VALID/DDIRTY` state exists, then its state is changed to `IN_USE` and `VALID/DIRTY`.

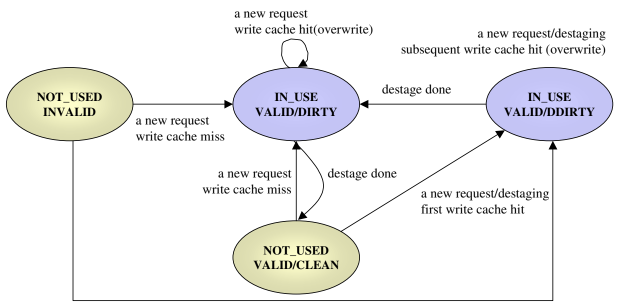


Figure 4. Stage diagram of cache entries within the write cache by the destage process

Next, we explain the destage process which flushes dirty write cache entries into physical disks in the background. Note that the old data and parity blocks are read into the read cache. However, those entries are not locked in the read cache until the computation of its new parity blocks completes. This implies that these blocks can be replaced with others before the new parity is calculated. Newly computed parity blocks are overwritten into the old parity blocks in the read cache, not the write cache. Also, the write of the new parity blocks are not delayed. In the presence of write cache overflows implying that no available cache entries exist for the newly arrived data from the host, subsequent write requests bypass the write cache, *i.e.*, the write requests are treated as in the traditional RAID5. Finally, host read requests are given a higher priority than destage requests. Then, the destage process starts when the current occupancy reaches its HWM and then stops when the current occupancy goes down to its LWM. The destage process first scans the write cache and then selects a dirty cache entry not currently in destaging. Then, it decides if the old data and parity blocks of the entry reside in the read cache. If not, it issues *destage read* requests for the missing blocks. After reading all the old data and parity blocks by the destage reads, it computes the new parity and generates two *destage write* requests for the new data and parity. Note that a possibility exists that the old blocks in the read cache can be replaced while the destage read requests of missing blocks are in progress. Thus, the destage process goes through another checking process to determine whether all the blocks are still there. When the destage write requests are completed, the state of the dirty cache entry is changed into *clean*, as previously described.

I/O Workload and Simulation Parameters: Basically, we employ synthetically generated I/O workloads throughout our performance evaluations. Each workload generator models an I/O application which issues an I/O request and then waits for its completion in a synchronous manner. We assume that each I/O application generates 8KB read or write requests with one percent of spatial locality, where the read and write ratio is set to 4:1. Using realistic I/O workloads is one of our future work.

Table 1 shows configuration parameters which are used in our cache RAID5 simulator. The simulated RAID system consists of five(5) disks, four of which are used for data and one for parity. Since the read cache is larger than the write cache, the size of the read cache is two times larger than that of the write cache. The cache entry size of each cache is set to 8KB.

In the experiments, the static cache occupancy levels for HWM and LWM are set to 0.7 and 0.3 of the whole write cache size, respectively. Note that these static threshold values are generally determined in an ad-hoc manner independently of a time-varying I/O workload. Recall that it is the main motivation of this paper. The configuration of the base increasing and decreasing rates of the cache occupancy levels takes the following two steps. First, the base decreasing rate μ_B^{max} was empirically obtained from our simulator under the condition that no intervention of host read requests exists. Thus, we obtained 70 IOPS with 8KB requests. Second, the inequality of Corollary 2 is applied to the obtained μ_B^{max} value in order to decide the value of λ_B. Our experiments takes the largest number of λ_B by multiplying $1 - \rho_L (= 0.7)$ with 70. Finally, the current changing rates of the cache occupancy level are monitored every 400 msec. Also, the independent disk features 512-byte sector, 1962 cylinders , 19 tracks per cylinder, 72 sectors per track, and 4002 rpm.

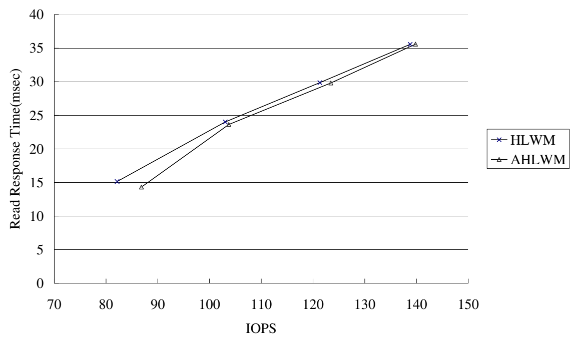
Table 1. Configuration parameters of our cached RAID5 simulator

Parameter	Value
RAID Size	4.19 GB
Number of Disks	5 (data:4, parity: 1)
Read Cache Size	16 MB (2048 cache entries)
Write Cache Size	8 MB (1024 cache entries)
Cache Entry Size	8 KB
Read/Write Ratio	4:1
ρ_H	0.7
ρ_L	0.3
μ_B	70 IOPS
λ_B	49 IOPS ($0.7\mu_B^{max}$)
Monitoring Interval	400 msec

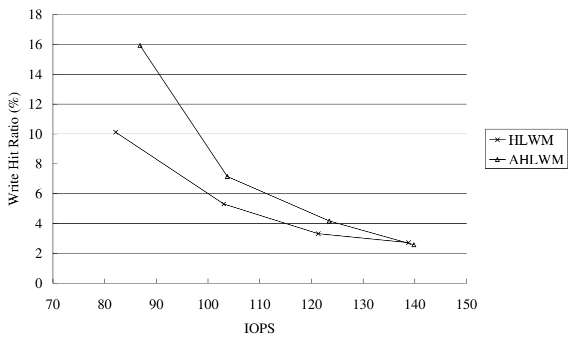
3.2. Simulation Results

The I/O performances of the HLWM and AHLWM algorithms are evaluated in terms of the response time of reads, write cache hit ratio, and disk utilization with an increase of I/O throughput. The response time of read requests has become more significant, because fast writes were achieved with non-volatile memory. This response time can be affected by disk utilization, especially the number of destage I/Os, and write cache overflows, as well as the read cache hit ratio. The high level of the write cache occupancy enhances the write cache hit ratio, resultantly reducing the response time of reads. Disk utilization is dependent mainly on the number of read and destage I/O requests. Of these, only destage I/O requests are directly related to an underlying destage algorithm.

I/O Performance with Different I/O Workloads: Figures 5–6 show the variation of read response time, write cache hit ratio, the number of disk I/Os issued by destage requests (briefly called destage I/Os), and disk utilization of the HLWM algorithm and the AHLWM algorithm with the increase of I/O throughput. Throughout all experiments, the base increasing rate of the cache occupancy λ_B and the base decreasing rate of the cache occupancy μ_B are set to $49(0.7\mu_B)$ IOs/sec and 70 IOs/sec. As shown in Figures 5–6, the AHLWM algorithm outperforms the HLWM algorithm. The performance differences are maximized especially under a light I/O workload in that its $\lambda(t)$ is relatively lower than the base value. As the I/O workload increases, however, the performances of both algorithms become almost equal, even though the AHLWM algorithm works slightly better than the HLWM algorithm. The performance



(a) Read response time



(b) Write cache hit ratio

Figure 5. Variations of read response time and write hit ratio of the HLWM algorithm and the AHLWM algorithm with the increase of I/O throughput : (a) read response time, (b) write cache hit ratio, where $\rho_H = 0.7$**,** $\rho_L = 0.3$**,** $\lambda_B = 49$ **IOs/sec, and** $\mu_B = 70$ **IOs/sec**

enhancements of read response time in the AHLWM algorithm are primarily due to time-varying, adaptive thresholds, *i.e.*, both HWM and LWM at each time are determined based on the current increasing and decreasing rates of the cache occupancy level. Under a light I/O workload, the AHLWM algorithm keeps up the cache occupancy level higher than the HLWM algorithm, meaning that the write cache hit ratio increases. It is easily inferred that the high level of the write cache hit ratio enhances the response time of reads *i.e.,* a high write cache hit ratio can reduce the number of destage I/Os, which then lessens disk utilization by destage I/Os.

A Variation of the Cache Occupancy Level: Figure 7 presents the average cache occupancy levels at each I/O request arrival rate of the previous results. It is clearly shown that the AHLWM algorithm maintains the cache occupancy level adaptively to the current I/O workload, *i.e.,* it keeps the level of the cache occupancy relatively high under light workloads, ranging from 82 to 140 I0s/sec. We can expect that HWM and LWM of the HLWM algorithm become similar to those of the AHLWM algorithm as the I/O workload increases. A series of subsequent results will prove this expectation. Also, we observed that the cache occupancy levels of both the HLWM and the AHLWM algorithms grow abruptly under heavy workloads beyond 140 IOs/sec, because the increasing rate of the cache occupancy level $\lambda(t)$ is almost equal to the decreasing rate of the cache occupancy level $\mu(t)$ under such a condition. Note that the write cache will be filled up shortly, if the $\lambda(t) > \mu(t)$ condition is sustained.

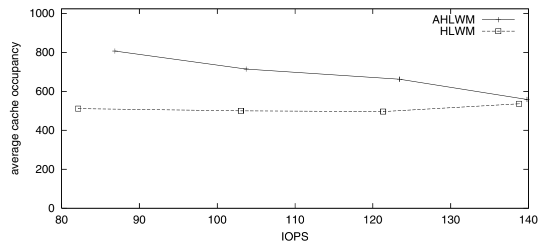
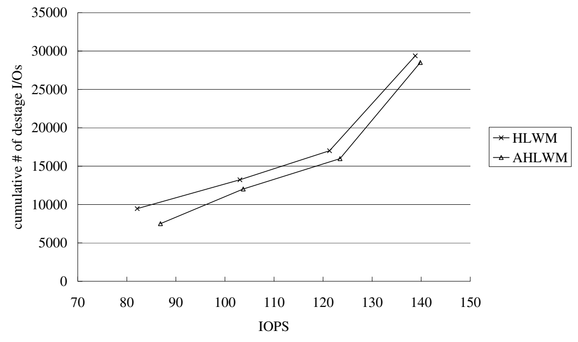
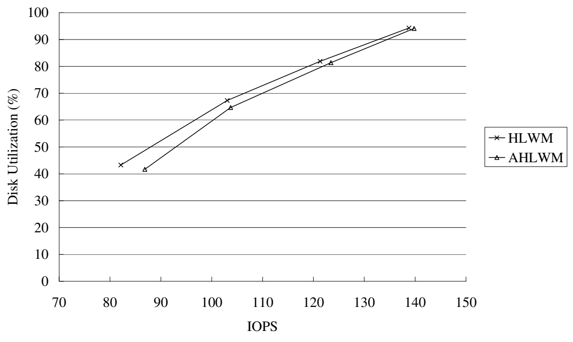


Figure 7. Average cache occupancy level of the HLWM algorithm and the AHLWM algorithm as a function of I/O throughput, where $\rho_H = 0.7$**,** $\rho_L = 0.3$**,** $\lambda_B = 49$ **IOs/sec, and** $\mu_B = 70$ **IOs/sec**

Finally, Figure 8 depicts the variation of the cache occupancy level of the HLWM algorithm and the AHLWM algorithm. The x-axis represents the current simulation time, and the y-axis, the current cache occupancy level. The figures also reveal that HWM and LWM of the AHLWM algorithm are very adaptive to various types of I/O requests, whereas those of the HLWM algorithm are time-invariant, irrespective of a varying I/O workload. Specifically, Figure 8(c) shows that HWM and LWM of the AHLWM algorithm is set higher than those of the HLWM algorithm under a heavy I/O workload.



(a) Cumulative number of destage I/Os



(b) Disk utilization

Figure 6. Variations of the cumulative number of destage I/Os and disk utilization of the HLWM algorithm and the AHLWM algorithm with the increase of I/O throughput : (a) cumulative number of destage I/Os, (b) disk utilization, where $\rho_H = 0.7$, $\rho_L = 0.3$, $\lambda_B = 49$ IOs/sec, and $\mu_B = 70$ IOs/sec

3.3. Discussion

We will briefly discuss an issue associated with a bursty I/O workload. In theory, the AHLWM algorithm is expected to be more tolerant of a burst of I/O requests than the HLWM algorithm, because it may detect the burst I/O in advance by tracing the current changing rate of the cache occupancy level, *i.e.*, when the current increasing rate of the cache occupancy level is higher than the base value, the AHLWM algorithm predicts that a flurry of I/O requests is near at hand, so the destaging process begins earlier than normal, thus avoiding a probable write cache overflow. In real experiments, however, the AHLWM algorithm made no difference in the level of burst tolerance, compared to the HLWM algorithm. Even in some cases, the AHLWM algorithm produced a greater number of cache overflows. Consider the case where the AHLWM algorithm keeps the level of cache occupancy high, along with a slow cache occupancy increasing rate followed by bursty write I/O requests. In this case, the AHLWM algorithm is more susceptible to write cache overflows than the HLWM algorithm.

To resolve this problem, we can exploit a marginal value to determine the value of HWM. We can modify Equation (9) to compute HWM at time k of $\acute{\rho}_H(t)$ as follows:

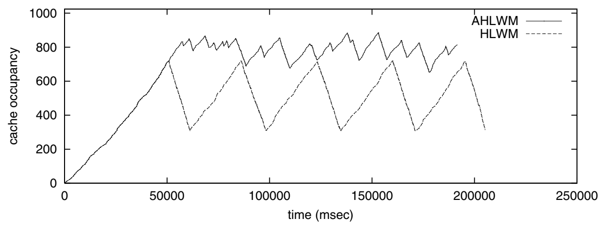
$$\acute{\rho}_H(t) \quad = \quad min\{S - \rho_{margin}, \rho_H(t)\}, \qquad (13)$$

where ρ_{margin}, $(0 \leq \rho_{margin} \leq S)$, represents a marginal threshold. Then, HWM is not set to higher than $1-\rho_{margin}$. As our future work, we need to justify that such a marginal threshold approach works effectively under a bursty I/O workload.

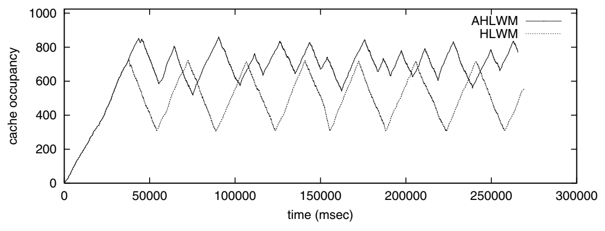
4. Conclusion and Future Work

In this paper, we have proposed an Adaptive High-Low Water Mark destage (AHLWM) algorithm for cached RAID5. While the destaging thresholds of previous destage algorithms – the High-Low Water Mark and Linear Threshold algorithms – are determined based only on the level of the current cache occupancy, those of the AHLWM algorithm are dynamically decided according to the changing rates of the level of the current cache occupancy. Performances of the HLWM algorithm and the AHLWM algorithm were evaluated in terms of the response time of reads, the write cache hit ratio, disk utilization, the number of destage I/Os, and average cache occupancy levels. We have shown that the AHLWM algorithm outperforms the HLWM algorithm under various workloads. It has also been shown that the AHLWM algorithm maintains the cache occupancy level adaptively to varying input requests, *i.e.,* while it keeps the cache occupancy level relatively high under light workloads, it lessens the level as the workload increases.

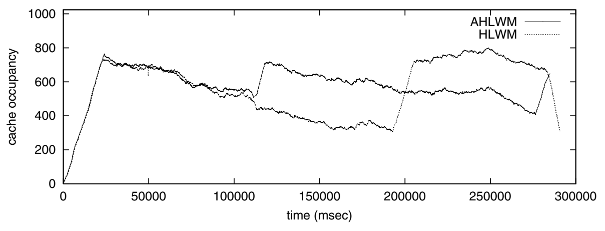
In subsequent research work, we need to explore the behavior of the AHLWM algorithm under much heavier and various types of I/O workloads, especially under bursty I/O requests. In addition, we plan to devise a systematic scheme to determine a pair of base increasing and decreasing rates of the cache occupancy level for a given underlying storage system. Finally, we will implement the AHLWM algorithm upon an actual RAID system called PosRAID [3] and then compare its performance with that of the HLWM algorithm.



(a) IOPS = 82



(b) IOPS = 103



(c) IOPS = 140

Figure 8. The variation of the cache occupancy level of the HLWM algorithm and the AHLWM algorithm as a function of IOPS, where $\rho_H = 0.7$, $\rho_L = 0.3$, $\lambda_B = 49$ IOs/sec, and $\mu_B = 70$ IOs/sec

Acknowledgments

The authors would like to thank the Ministry of Education of Korea for its financial support through its BK21 program.

References

[1] J. Menon and J. Cortney. The architecture of a fault-tolerant cached raid controller. In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 76–86, May 1993.

[2] J. Menon and *et. al.* Floating parity and data disk arrays. *Journal of Parallel and Distributed Computing*, pages 129–139, 1993.

[3] Y. Nam and *et. al.* Design and implementation of a high-performance raid system software for mid-range storage servers. *Technical Report CSE-SSL-2000-05, POSTECH*, August 2000.

[4] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks(raid). In *Proceedings of IEEE COMPCON*, pages 112–117, Spring 1989.

[5] M. Rosenblum and J. Ousterhout. The design and implementation of a log-structured file system. *ACM Trans. Computer Systems*, 10(1):206–235, August 1994.

[6] D. Stodolsky, G. Gibson, and M. Holland. Parity logging: Overcoming the small write problem in disk arrays. In *Proceedings of the 20th International Symposium on Computer Architecture*, May 1993.

[7] A. Varma and Q. Jacobson. Destage algorithms for disk arrays with nonvolatile caches. *IEEE Transactions on Computers*, 47(2):228–235, February 1998.