This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2014.2298394, IEEE Computer Architecture Letters

Synchronous I/O Scheduling of Independent Write Caches for an Array of SSDs

Junghee Lee*, Youngjae Kim†§, Jongman Kim*, Galen M. Shipman†
*Georgia Institute of Technology, †Oak Ridge Natinoal Laboratory
Email: {jlee36, jkim}@ece.gatech.edu, {kimy1, gshipman}@ornl.gov

Abstract—Solid-state drives (SSD) offer a significant performance improvement over the hard disk drives (HDD), however, it can exhibit a significant variance in latency and throughput due to internal garbage collection (GC) process on the SSD. When the SSDs are configured in a RAID, the performance variance of individual SSDs could significantly degrade the overall performance of the RAID of SSDs. The internal cache on the RAID controller can help mitigate the performance variability issues of SSDs in the array; however, the state-of-the-art cache algorithm of the RAID controller does not consider the characteristics of SSDs. In this paper, we examine the most recent write cache algorithm for the array of disks, and propose a synchronous independent write cache (SIW) algorithm. We also present a pre-parity-computation technique for the RAID of SSDs with parity computations, which calculates parities of blocks in advance before they are stored in the write cache. With this new technique, we propose a complete paradigm shift in the design of write cache. In our evaluation study, large write requests dominant workloads show up to about 50% and 20% improvements in average response times on RAID-0 and RAID-5 respectively as compared to the state-of-the-art write cache algorithm.

Index Terms—Redundant Array of Independent Disks (RAID), Solid-State Drive (SSD), flash memory, I/O scheduling, Write cache.

1 Introduction

In HPC storage systems, multiple disk drives are employed to build arrays of disk drives. Redundant arrays of inexpensive disks (RAID) were introduced to increase the performance and reliability of disk drive systems. RAID provides parallelism of I/O operations by combining multiple inexpensive disks, thereby achieving higher performance and robustness. RAID has become the de facto standard for building high performance and robust storage systems. Unlike in-place update operations in HDDs, SSDs incorporate software to allow out-of-place update operations and to map sectors from the host into their current locations in the SSDs. This out-of-place update operation eventually requires a sweep of storage area to find stale data and consolidate active data in order to create free space. This process, known as garbage collection (GC), can increase the service time of incoming requests significantly.

When SSDs are configured in RAID, the performance variability of individual SSDs becomes a major concern, because the overall performance of the SSD-based RAID could be limited by the slowest SSD. Even though several studies addressed the concern of the performance variability on a single SSD due to the GC operation, there is still a concern on the performance variability on an array of SSDs. In this paper, we investigate a solution on a RAID controller that can alleviate the performance degradation concern in the RAID of SSDs. We suggest a new *write cache architecture* on a RAID controller and a *request scheduling approach* to improve the overall I/O performance by overcoming the aforementioned problems. Our new write cache architecture maintains multiple sub-buffers for every SSD, and the request scheduler reorders the requests in their corresponding queues to increase the chance that multiple SSDs trigger GC at the same time, resulting in improved I/O performance. Also a *pre-parity-computation* technique for the RAID configuration with parity computation will help flexible rescheduling.

§*Corresponding author*
Manuscript submitted: 31-Aug-2013. Manuscript accepted: 30-Nov-2013. Final manuscript received: XYZ

2 Background

The write cache on the RAID controller employs non-volatile memory or battery-back memory [1], [2] in order to prevent data loss in case of power-failure on the system. As long as there is free space in the write cache, an incoming write request is stored in the write cache and it is immediately committed to the requester. Otherwise, the request should be pending in the I/O queue until the write cache becomes available. The data in the write cache will be synchronized with the disks later in the background. This process is called *destaging* and the write operations to the disk issued for destaging are called *destage writes*. A destage write is split into multiple *strips*. Depending on the RAID configuration, a parity strip may be added. Each strip is stored in its corresponding disk. The strips and the parity strip issued for a destage write are called *stripe*. The write requests belonging to the same stripe are called *write group*.

To fully exploit benefits of a write cache, a cache controller should be designed to leverage temporal and spatial locality as well as to resist bursty writes [1], [2]. High temporal locality can be achieved by keeping as much hot data as possible. On the other hand, to sustain bursty writes, dirty data in the cache should be destaged in advance before the bursty writes come in. Otherwise, it will increase pending requests in the queues, causing delays for writes. However, early destaging will redduce the chance of coalescing writes on writes, reducing temporal locality. In the context of HDDs, leveraging spatial locality means minimizing mechanical movement inside a HDD. Extensive research has been conducted on these cache scheduling algorithms to exploit the spatial locality on a HDD [3]. In order to minimize the delay due to the cache being full while maintaining locality requirements, the cache controller should be carefully designed to make a smart

1556-6056 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2014.2298394, IEEE Computer Architecture Letters

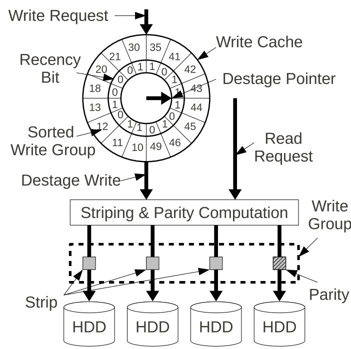


Fig. 1. A typical RAID controller architecture employing a write cache in WOW [1].

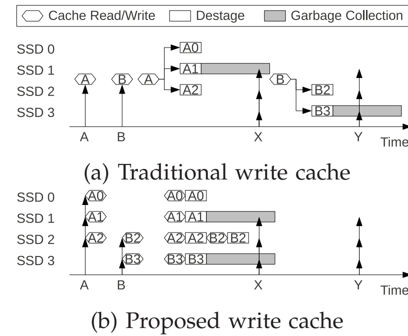


Fig. 2. Timing diagram of processing write requests A and B and read requests X and Y in an array of 4 SSDs.

decision on when to destage, what to destage, and how much to destage.

For the RAID controllers, WOW [1] has been proposed to exploit both spatial locality and temporal locality at the same time. One step further, STOW [2] enhances WOW by separating random writes from sequential writes. The data structures of the write cache are illustrated in Figure 1. Even if WOW [1], and STOW [2] significantly improve write performance, their work is limited only to an array of hard drives. On the contrary, our work, more importantly, focuses on the design of write cache for an array of SSDs. We consider the internal characteristics of SSDs with more diverse resources than HDDs, such as multiple chips, channels, multiple cores, etc. To the best of our knowledge, no prior work has been performed on the write cache design for *an array of SSDs*. GC needs to be carefully considered in cache management on the RAID controller for SSDs.

3 SSD-AWARE WRITE CACHE DESIGN

Wise ordering for Writes (WOW) [1]: Figure 1 shows an overview of the WOW cache algorithm. A write hit can be either a hit on a *strip* or a hit on a *write group*. A write group consists of multiple strips. A hit on a strip means there already exists the requested strip in the write cache. Even if the requested strip does not exist, its write group could exist (hit on a write group). Since a parity strip is computed for each write group and parity is computed when a write group is destaged, a hit on a write group could reduce the number of parity computations. Write groups are always ordered by their logical block address (LBA), which leverages spatial locality. When destaging is necessary, the destage pointer advances. How to determine when to destage will be discussed shortly. If the recency bit of the write group pointed by the destage pointer is zero, the write group is destaged. If the recency bit is one, the write group is retained and the recency bit is cleared. Then the destage pointer advances again until it finds a write group whose recency bit is zero. The recency bit is set when the write group is hit. The recency bit gives one more chance for hit write groups to survive for one more cycle, which leverages temporal locality. WOW adopts a linear threshold scheduling [4] to determine when to destage, where two thresholds are involved: low and high thresholds. Destaging begins to be triggered when the number of write groups (N) in the write cache reaches a low threshold. The destaging rate linearly increases as N increases. If N reaches a high threshold, it runs at a full rate.

Synchronous Independent Write Cache (SIW): All write back cache algorithms including WOW, however well designed, does not consider the characteristics of drives, and will suffer from performance degradation due to uncoordinated garbage collection processes on an array of SSDs. Therefore, *our claim is that write cache algorithms for an array should be completely redesigned by considering the device characteristics.*

The main problem of applying traditional write cache architecture to an array of SSDs is that the destage write is bounded by the slowest SSD. This problem is not addressed by the state-of-the-art write cache designs such as WOW [1] and STOW [2]. We empirically observed that GC runs for 3 ms on average, while a write operation in an SSD takes 0.2 ms [5]. It would be very inefficient if the cache controller cannot proceed to the next destage write for more than 3 ms because of a single SSD delayed by GC, even though all other strips can be processed within 0.2 ms. Figure 2 shows this situation in an example.

In Figure 2(a), suppose that write requests A and B arrive, which are stored in the cache immediately, and the write cache controller decides to destage at some later time. Request A is split into A0, A1 and A2 which are going to SSD0, SSD1 and SSD2, respectively. If a strip that goes to SSD1 is delayed by GC, the destage write cannot be committed until SSD1 finishes GC and processes strip A1. Only after SSD1 finishes processing A1 can the cache controller destage the next write group. If bursty write requests come during this period, the write cache may become full, which incurs long latency to subsequent write requests. Even while SSD1 is delayed by GC, other SSDs can accept strips. If the cache controller can issue the next destage writes to other SSDs, it would reduce the chance of the write cache being full, which results in more efficient use of write cache space. This observation constitutes the primary motivation of our proposed technique, which is described in the rest of this section.

The proposed design of Synchronous Independent Write Cache (SIW) is described in Figure 3. The main difference from WOW or the traditional architecture (illustrated in Figure 1) is that multiple write caches are employed. They are independent of one another in that strips belonging to one write group do not have to be destaged together because a parity strip is already computed. The differences are summarized as follows: First, SIW employs as many write caches as the number of SSDs in the array. Thus, an entry of each write cache is a *strip*. Second, the destages from write caches are synchronized. Third, the parity can be computed in advance before requests are stored in the write cache, which we call pre-parity-computation. Even if any SSD is delayed by GC, independent scheduling of write caches allow other SSDs to accept a strip from their corresponding write caches.

The write caches are synchronized in that they attempt to destage together at the same time. If all the write caches have at least one strip to destage, they destage together. Note that those

1556-6056 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2014.2298394, IEEE Computer Architecture Letters

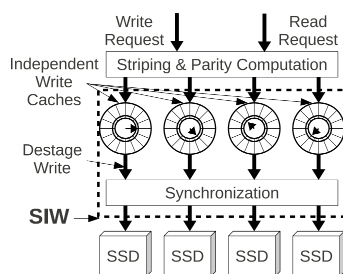


Fig. 3. An architectural overview of SIW.

strips destaged together do not have to belong to the same write group. The destage pointer advances independently, it advances while clearing the recency bit, until it finds a strip whose recency bit is already cleared. The strip with cleared recency bit is ready to destage. When the pointer finds a strip to destage, it has to stop there until all the other pointers find one. However, if the waiting time of those strips waiting for being destaged together exceeds a predefined threshold, it is allowed to destage to prevent the waiting write caches from becoming full. Except for those having no strip, the other write caches destage together.

The synchronization mechanism contributes to the performance improvement by increasing the chance of GCs running simultaneously. In the example of Figure 2(b), GC of SSD1 and SSD3 run together by the synchronization. Let us suppose read requests X and Y come, both of which span to SSD1, SSD2 and SSD3. In Figure 2(a), both X and Y are delayed by GC whereas in Figure 2(b) Y is not. As illustrated in this example, the synchronization mechanism broadens the time window when no GC is running. If the write caches are placed before parity computation, they cannot work independently, which limits the efficacy of synchronization.

We propose a *pre-parity-computation technique*, which computes the parities of blocks before they are placed in their caches. Let us suppose A2 and B2 are parity strips of requests A and B respectively. A0, A1, and A2 should be destaged together because A2 is computed while request A is split into A0 and A1. Even though SSD3 is available, B3 cannot be destaged because of the constraint that B2 and B3 should be destaged together. The pre-parity-computation allows write caches to work independently, which gives better chance for synchronization.

The limitation of the pre-parity-computation is that the delay taken by parity computation is always added to the response time of any requests. In contrast, if the write queues are placed before parity computation, the delay is hidden when the cache hits. However, this overhead can be minimized by improvement from the GC synchronization, as will be demonstrated by experiments. The proposed design has no issues with recovery from a disk failure because it employs non-volatile memory as a cache. If the RAID controller detects failure on an SSD, it destages all the data stored in the cache into the SSDs except for the failed SSD before it starts rebuilding the failed disk. Then it can rebuild the failed SSD based on other SSDs since all the up-to-date data is in the SSDs.

Even though the pre-parity-computation incurs an overhead of parity computation for every request, it still exploits the temporal and spatial locality. In SIW, every write request incurs parity computation, but if its subrequests are found in the write caches, they do not incur additional I/Os to SSDs.

4 EVALUATION

Environment: In order to study the performance implications of our proposed scheme, we enhanced the DiskSim and SSD simulator developed by Microsoft Research [5]. Our tests were performed on RAID-0 and RAID-5. In RAID-5, 8 SSDs are for data strips and one is for a parity strip whereas in RAID-0, all of 8 SSDs are for data strips. The strip unit size for RAID is 128KB, so the 1 MB request constitutes a write group (128 KB $\times$ 8 = 1 MB) for RAID-0 configuration. In simulating an SSD, we used 15% reserved free blocks with 5% minimum free blocks, and a greedy GC policy. We also evaluated with a semi-preemptive GC scheme [6]. Each SSD uses four flash chip packages, where each package consists of four planes, and each plane uses 512 blocks. Each block consists of 64 4KB pages. Thus, the size of each SSD is 2GB. Page read and write times of 0.025ms and 0.2ms are used and block erase time of 1.5 ms is used.

TABLE 1
Varying workload parameters. W(R, I, W) denotes request size R (KB), inter-arrival time I (ms), and write percentage W (%).

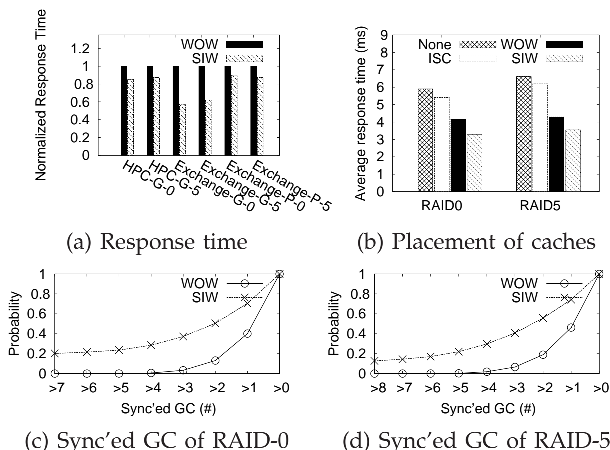
Parameter	Values
Request size (R)	4 KB, 512 KB, 1024 KB, 4096 KB
Inter-arrival time (I)	10 ms, 20 ms, 40 ms
Write percentage (W)	20 %, 60 %, 80 %

We use a mixture of real-world and synthetic traces to study the impact of our proposed scheme on a wide spectrum of HPC workloads. For the realistic HPC workload, we used a synthesized trace that was generated based on the study of real I/O workload characterization for one of the largest storage systems for HPC [7]. Kim et. al. [7] developed synthesized workloads for HPC, where it shows about 60% writes and 40% reads, a bi-normal distribution for request size, and a poisson distribution for inter-arrival times with an average of 20ms. For the request size distribution, there are 50% 4K small requests, and 50% large requests (17% of 512KB and 32% of 1MB requests). We also widen the spectrum of our investigation using synthetic workloads by varying various workload properties, namely I/O request size, arrival rate of the requests, and the percentage of write requests. Table 1 summarizes the values of these parameters used in the experiments. In addition, we use Microsoft (MSR) Exchange Server traces as a real-world workload [8]. The MSR workload requires 4TB storage. So, we grouped the traces in multiple sub-traces each of which can be fed into our storage simulator. Due to the page limit, we only show results of two traces. However, other traces show similar findings.

Results: Figure 4 shows the effectiveness of our proposed idea, SIW over WOW for HPC and Exchange workloads. The notion of X label is *[workload]-[GC policy]-[RAID configuration]*. G indicates the greedy GC and P does semi-preemptive GC. Zero (0) and five (5) of the RAID configuration mean RAID-0 and RAID-5, respectively. In Figure 4(a), we observe the improvement of response times by SIW over WOW is by up to 42% . For RAID-0, where there is no pre-parity-computation overhead, the improvement is 14.74% for HPC and 42.62% for Exchange, which is mainly due to GC synronization. In the RAID-5 configuration, even though the pre-parity-computation incurs 5.21% overhead in response times, this computation penalty is compensated by the increased efficiency of SIW. As a result, the improvement by SIW for RAID-5 is slightly less than that by WOW for RAID-0, but we can still observe the improvement in the response times by 12.72% and 37.93% for HPC and Exchange, respectively. Since Exchange workload exhibits frequent and large requests, the performance improvement is larger for Exchange than HPC. With a semi-preemptive GC

1556-6056 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/LCA.2014.2298394, IEEE Computer Architecture Letters



(a) Response time (b) Placement of caches

(c) Sync'ed GC of RAID-0 (d) Sync'ed GC of RAID-5

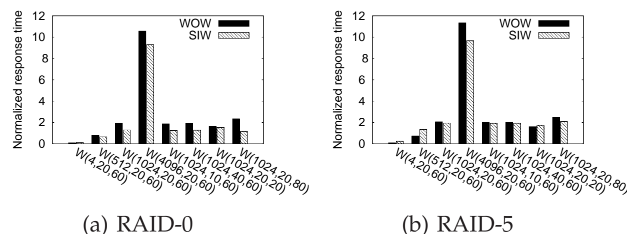
Fig. 4. (a) compares the performance of WOW and SIW for HPC and Exchange workloads. (b) compares the placement of write caches. (c) and (d) show the probability of GC synchronization for and RAID-0 and RAID-5 respectively. For (b),(c) and (d) we used HPC workloads.

scheme [6], we observed up to 12.70% improvement for the Exchange workload.

Figure 4(b) demonstrates that it is more efficient to place the write cache on the RAID controller (WOW and SIW) than in individual SSDs. In "ISC" configuration of this experiment, individual SSD has its own write cache and the total amount of write cache is the same with WOW and SIW. "None" denotes SSDs without write cache.

The synchronization mechanism improves the performance by increasing the probability of GCs operating simultaneously. Figure 4(c) shows the cumulative distribution function (CDF) of the number of synchronized GCs for RAID-0. We count the number of SSDs that run GCs during a fixed time slice, for which we use 0.1 ms in our measurement. In WOW, the probability of the number of synchronized GC being at least two, $P(n \geq 2)$, is 0.40. By employing the synchronization mechanism, the probability is drastically increased. $P(n \geq 2)$ becomes 0.70 (75.00% improvement). Figure 4(d) shows that RAID-5 exhibits the same trend along with RAID-0. $P(n \geq 2)$ is increased from 0.46 to 0.74 (60.86% improvement).

In order to cover a wide-range of workload characteristics, we evaluated our design with synthetic workloads. Figure 5 shows the results comparing SIW and WOW. Figure 5(a) shows the results of RAID-0. The main benefit of employing SIW comes from large requests. Compared to WOW, the response time of W(512, 20, 60), W(1024, 20, 60), and W(4096, 20, 60) is reduced by 16.26%, 32.51%, and 12.05%, respectively. Since the response time of large requests is much longer than that of small requests, the response time reduction for large requests substantially improves the overall performance. When the request size is small (e.g., W(4, 20, 60)), the overhead of the pre-parity-computation is not compensated. Even though the parity computation is not required for RAID-0, the overhead of striping still exists because every write request requires striping in SIW whereas it may not be required upon a cache hit in WOW. However, although the response time is increased by 12.05%, the absolute value of the increased response time is as small as 0.01 ms because the response time for a small request is very short. Thus, it does not have significant impact on the overall performance. Varying inter-arrival time (e.g., W(1024, 10, 60) and W(1024, 40, 60)) and percentage of write requests (e.g., W(1024, 20, 20) and W(1024, 20, 80)) does not affect the overall trend. If the RAID configuration requires parity com-



(a) RAID-0 (b) RAID-5

Fig. 5. Performance comparison for synthetic workloads.

putation (RAID-5), the overhead of the pre-parity-computation increases. Figure 5(b) shows the results of RAID-5. For W(1024, 20, 60), W(4096, 20, 60), the performance improvement is 5.79% and 14.79%, respectively. For small requests (e.g., W(4, 20, 60)), the response time increases sharply in terms of percentage, but the absolute value of their increment is still as small as 0.14 ms.

b In summary, the extra overhead of SIW includes rescheduling and pre-parity-computation overheads. The rescheduling overhead is incurred by the synchronization because a write cache may need to wait until other write caches get to have one strip to destage. The pre-parity-computation overhead includes the striping and parity computation overheads, as discussed above. For large requests, these overheads are compensated and we can observe significant performance improvement, but they are not compensated for small requests. Based on this result, we envision to develop a hybrid approach that switches between WOW and SIW adaptively to the workload.

5 Conclusion

This paper proposes a synchronous write cache algorithm with a pre-parity-computation technique for an array of SSDs. Unlike a traditional write cache, the proposed architecture employs as many separate write queues as the number of SSDs in the array. By employing multiple write queues, an impact of GC on any write queue can be isolated from other queues. The results revealed that the synchronization of destage writes across SSDs can increase the probability of overlapped GC operations, improving the overall performance of an SSD RAID.

Acknowledgment

This research was supported in part by, and used the resources of, the Oak Ridge Leadership Computing Facility, located in the National Center for Computational Sciences at ORNL, which is managed by UT Battelle, LLC for the U.S. DOE (under the contract No. DE-AC05-00OR22725)

References

[1] B. S. Gill and D. S. Modha, "WOW: wise ordering for writes - combining spatial and temporal locality in non-volatile caches," ser. FAST'05.

[2] B. S. Gill, M. Ko, B. Debnath, and W. Belluomini, "STOW: a spatially and temporally optimized write caching algorithm," ser. USENIX'09.

[3] T. R. Haining, "Non-volatile cache management for improving write response time with rotating magnetic media," Ph.D. dissertation, University of California, Santa Cruz, 2000.

[4] A. Varma and Q. Jacobson, "Destage algorithms for disk arrays with nonvolatile caches," *Computers, IEEE Transactions on*, vol. 47, no. 2, pp. 228 –235, feb 1998.

[5] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," ser. USENIX'08.

[6] J. Lee, Y. Kim, G. Shipman, S. Oral, and J. Kim, "Preemptible I/O scheduling of garbage collection for solid state drives," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 32, no. 2, pp. 247–260, 2013.

[7] Y. Kim, R. Gunasekaran, G. M. Shipman, D. A. Dillow, Z. Zhang, and B. W. Settlemyer, "Workload characterization of a leadership class storage," ser. PDSW'10.

[8] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating server storage to ssds: analysis of tradeoffs," ser. Eurosys'09.

1556-6056 (c) 2013 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.