

BPAC: An Adaptive Write Buffer Management Scheme for Flash-based Solid State Drives

Guanying Wu, Ben Eckart, Xubin He
Department of Electrical and Computer Engineering
Tennessee Technological University
Cookeville, TN 38505
Email: {gwu21, bdeckart21, hexb}@tntech.edu

Abstract—Solid State Drives (SSD’s) have shown promise to be a candidate to replace traditional hard disk drives, but due to certain physical characteristics of NAND flash, there are some challenging areas of improvement and further research. We focus on the layout and management of the small amount of RAM that serves as a cache between the SSD and the system that uses it. Of the techniques that have previously been proposed to manage this cache, we identify several sources of inefficient cache space management due to the way pages are clustered in blocks and the limited replacement policy. We develop a hybrid page/block architecture along with an advanced replacement policy, called BPAC, or Block-Page Adaptive Cache, to exploit both temporal and spatial locality. Our technique involves adaptively partitioning the SSD on-disk cache to separately hold pages with high temporal locality in a page list and clusters of pages with low temporal but high spatial locality in a block list. We run trace-driven simulations to verify our design and find that it outperforms other popular flash-aware cache schemes under different workloads.

I. INTRODUCTION

Solid state drives (SSD’s) are set to supplant traditional hard disk drives (HDD’s) in nearly every domain of storage computing, from server applications to home desktops to MP3 players. These drives are predominantly made from banks of NAND flash memory, and though significantly different from platter drives, are exported to the OS as simple block devices. As prices have dropped over 100x in the last 5 years [1], consumer interest is growing in many markets since SSD’s have many inherent benefits over HDD’s.

The swift and almost inevitable rise to ubiquity notwithstanding, SSD’s do suffer from several performance quirks arising from the physical nature of NAND flash and architectural constraints of their controllers. The most notable problems include: the inability to modify data in-place, read/write performance asymmetry, and slow and constrained erase functionality. There have been a wealth of techniques developed to circumvent these issues, including work at the Flash Translation Layer (FTL), new caching mechanisms, and new ways to exploit the parallelism of the flash device. Our work presented in this paper falls into the area of caching. Among the various *flash-aware* disk cache management schemes in the literature, BPLRU [2], FAB [3], and CLC [4] are the most representative. By carefully examining the ways these schemes organize data structures in the cache and the schemes they use to perform cache replacement, we found that there exist a few problems that may cause inefficient use of the cache space, which can result in unnecessary cache destages.

In this paper, we present a novel cache management scheme for buffering *write* requests to the SSD in the small RAM portion of the drive. We identify critical drawbacks of existing write buffer policies, and as our approach to solve this problem, we propose a new data structure for the buffered pages, as well as a novel mechanism that can estimate the spatial locality of the workload and make decisions to replace cache elements of low spatial locality.

Specifically, we create a new cache data structure, *Dual-list*, that partitions the SSD on-disk cache space into a page-based list and a block-based list for the purpose of buffering write requests, and we show that this particular architecture more efficiently uses cache space compared with the pure block-based list. We develop a new metric (BIRD) for flash-based SSD to evaluate the spatial locality of the disk I/O workload, based on which we propose an approach adaptive to different workloads to dynamically differentiate the low spatial locality clusters from high spatial locality ones. In addition, we develop a replacement policy that makes differential treatments based on access patterns. We compare our algorithm with other common flash-aware cache algorithms using trace-driven simulations and show that BPAC, or Block-Page Adaptive Cache, is superior under most real-world traces we used.

The rest of the paper is organized as follows: Section II gives a brief overview of the major softwares used in SSD’s. In Section III, we discuss the potential problems of previous flash-aware cache schemes. In Section IV, we discuss the design of BPAC in details. We analyze its performance with respect to other schemes in Section V, and conclude with our final comments in Section VI.

II. BACKGROUND AND RELATED WORK

SSD’s have inherent drawbacks resulting from NAND flash architecture, particularly the slow erase times at block-level granularity, lack of overwrite capabilities, read/write asymmetry (e.g., as shown in Table I), and wear-out from repeated accesses. To address these challenges, there have been many approaches working on *garbage collection process*, *wear-leveling*, *logical to physical mapping scheme*, etc., as well the *on-disk cache management scheme*. Two of these important techniques, FTL and flash-aware cache schemes, are described in next two subsections.

TABLE I
VALUES FROM [11] FOR A SAMSUNG 4 GB FLASH MODULE.

Page Read to Register	25 μ s
Page Program from Register	200 μ s
Block Erase	1.5 ms
Serial Access to Register	100 μ s

A. Flash Translation Layer

An SSD exports itself as a block device by adopting a software layer called the Flash Translation Layer (FTL) in between the host interface and the raw flash memory. Two functionalities of FTL, the mapping scheme and the garbage collection process, which are most related with on-disk buffer scheme, are discussed below.

1) *Mapping Schemes*: The mapping schemes of FTL's can be classified into two types: page-level mapping, with which a logical page can be placed onto any physical page; or block-level mapping, with which the logical page LBA is translated to a physical block address and the offset of that page in the block. *Log-block FTL's* [6] reserve a number of physical blocks that are not externally visible for logging pages of updated data. According to the block association policy (how many data blocks can share a log block), there are mainly three schemes, BAST [7], FAST [8], and SAST [9]. Compared to the above logging schemes, which solely work on the flash memory, a recent approach [10] has been proposed to take advantage of the in-place update capability of the PRAM, where the PRAM is used for the log region and the flash for the data region.

2) *Garbage Collection Process*: For log-block FTL's, when free log blocks are not sufficient, the *garbage collection* process (merge operation) is executed, which merges clean pages on both the log block and data block together to form a data block full of clean pages. According to the distribution of clean pages, there are three types of merge operations, *full merge*, *switch merge*, and *Partial merge* [8].

B. Flash-Aware Cache Schemes

To address the complexity of the FTL as well as the read/write asymmetry and the erase penalty, an on-disk cache is needed that can absorb repeated writes and reform random workloads and export them to the FTL as sequential.

BPLRU or block-level LRU [2]: This scheme is proposed to exploit the spatial locality of the workload by grouping pages that belong to the same *data blocks* (the following two schemes do this as well) into *page clusters* and ordering the clusters in an LRU fashion; the recency of a cluster depends on the most recently accessed page of the cluster. The LRU cluster is selected as the replacement victim. *FAB or Largest Cluster* [3]: This scheme maintains a list of page clusters sorted by their size in the cache, and the largest cluster is always the replacement victim. *CLC or Coldest and Largest Cluster* [4]: it is a mixture of block-level LRU and FAB: the cluster list is manually partitioned into a "size-independent" region for clusters of high locality, which are ordered in

an LRU fashion; a "size-dependent" region of low locality clusters ordered by the size. Clusters get evicted only if they are the largest of the clusters in the size-dependent region. The ratio of the number of clusters in the size-independent region to the total number of clusters is denoted as α . For comparison purpose, these same terms are used in this paper. *CFLRU* [12]: Clean-first LRU tries to leverage the read/write asymmetry by picking out pages to evict which are not dirty. Thus, the eviction will not lead to any actual write to the drive. *Griffin* [13]: Griffin is proposed to use a HDD as a write cache for SSD. By converting the update writes into a HDD-based log, which is eventually merged with the data on the SSD, Griffin takes advantage of HDD's high sequential write speed and large capacity to reduce the amount of writes directly serviced by the SSD while improve the sequentiality of the workload.

III. MOTIVATION

In this section, we discuss the potential problems of the popular flash-aware on-disk cache schemes. We find most problems are rooted in the ways that locality is exploited.

Pure block-based list: With this data structure, both temporal and spatial localities determine the recency of clusters. Consider a cluster containing a small number of hot pages and mostly cold ones (for single pages, hot/cold refer to the temporal locality). The spatial locality causes pages to be grouped together, while the temporal locality, which is represented by the repeated accesses to the hot pages, will keep updating the recency of a cluster. Thus, the cold pages will stay in the cache with the hot ones if they belong to the same block, causing a waste of cache space.

Early eviction: One major problem of FAB is that it does not protect the clusters that have high temporal or spatial locality from being evicted. Take the spatial locality for an example: in a sequential access stream {64, 65, 66,, 127}, assuming a data block is 64 pages (we hold this assumption all through this paper), a cluster of block #1 is formed; if this cluster is evicted before the page 127 is added-in, one new cluster of the same block will be formed by the rest pages. The FTL may log the early-evicted cluster in one log block, and if the garbage collection process is executed on this log block before FTL logs the subsequently formed cluster in the same log block, a costly partial or full merge is inevitable.

Efficiency of the LRU replacement policy: Considering the LRU-order cluster list (BPLRU), the set of clusters in the least recently used region (tail) may be assumed to have the same "coldness" and the temporal and spatial locality can be considered minimal. Thus, selecting the largest one in such a region can make more available cache space than selecting the cluster on the exact end of the list.

Non-adaptiveness in the partitioning scheme: By experimental trials with the workload, the CLC scheme finds out the optimal proportion $(1 - \alpha)$ of the size-dependent region upon which the *largest cluster* policy is applied; and the tuned α remains invariant throughout the workload. As said in [4], accurately identifying the *size-dependent* region is crucial: if

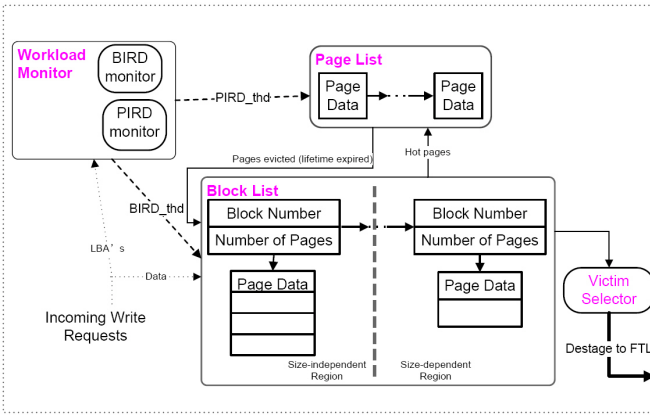


Fig. 1. BPAC Overview

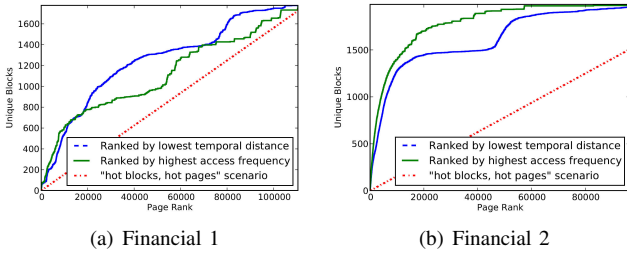


Fig. 2. Number of unique blocks vs. ranked pages. Hottest pages reside in many blocks, shown by the steepness of the curve at the leftmost part of the graph. Cold blocks are spread out over the same blocks that contain hot pages, shown by the flat regions on the middle and rightmost parts of the graph.

its size is smaller than the optimal, CLC regresses towards BPLRU; if its size is larger than the optimal, CLC will have the early eviction problem as FAB does.

IV. BPAC ARCHITECTURE AND DESIGN

In this section, we propose a hybrid adaptive write buffer management scheme for SSD called BPAC (Block-Page Adaptive Cache) to solve the problems of the legacy schemes.

A. Workload Analysis and the Motivation for Dual-List

We wish to know if clustering solely by block, as in BPLRU, leads to poor cache utilization. If a cold page gets accessed while a hot page from the same block is in the cache (or vice versa), the cold page has the danger of being “dragged along” by the nature of pure block-based caches. For our analysis, we look at two popular traces, Financial 1 and 2 (F1, F2) from the UMASS Trace Repository [14], which are large OLTP traces having write instructions on the order of 1 million.

We decided to investigate how many unique blocks the hottest pages use up. If hot pages reside in otherwise cold blocks, then the hottest pages will be spread out over many different blocks. If not, relatively few blocks will house the hottest pages. Thus, if we graph how many unique blocks are present for the x hottest pages, we can get an accurate depiction of how spread out the heat is, and if clustering by block will keep unwanted cold pages in the cache. From Figures 2(a) and 2(b), we can see that hot pages are spread

out among many different blocks. Grouping by block only will bring along unwanted cold pages, since very few hot pages actually reside in each block. It is clear that a scheme is needed that can detach hot pages with cold pages yet still evict by cluster.

B. BPAC Data Structure: the Dual-list

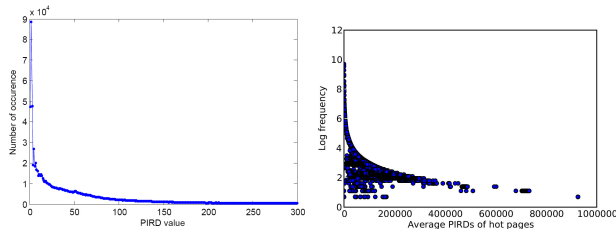
In BPAC, the cache space is partitioned into two parts, as shown in Figure 1, one for the p-list consisted of single pages, and the other for the b-list of page clusters. P-list serves as the repository of pages with high temporal locality (hot) ordered in an LRU fashion; b-list holds pages with low temporal locality (cold). The b-list is further divided into size-independent (clusters with high spatial locality) and size-dependent (clusters with low spatial locality) regions, in which the positions of clusters are determined by recency and size, respectively. Due to the fact that new incoming pages’ (cache misses) “hotness” is unknown, they are first accommodated in the b-list, in which pages are always merged together into a cluster if they belong to the same data block. It is commonly held that if a page is accessed more than once, its temporal locality tends to be much higher than pages accessed only once [15]. Thus, in our design we consider the “second” access (the first hit) as a sign of hotness. If a particular page in the b-list is accessed again, this page is moved to the p-list. When a page’s “lifetime” of staying in the p-list (the mechanism determining the lifetime is discussed in the next part) expires, it is moved back to the b-list.

C. Flash-aware Locality Metrics

For SSD’s, assuming a block-level mapping scheme common to most FTL’s, the spatial locality of page accesses on different data blocks can be considered to be minimal. Instead, *spatial locality* is about how close in virtual time the references are to the *unique* pages in the same data block. We refer to this block-level inter-reference distance as *BIRD*. For example, if two consecutive references to block #0 happens at time 10 and 34, respectively, then this BIRD is $(34 - 10 - 1) = 13$. To measure *temporal locality*, the traditional inter-reference gap or IRG is used, but for comparison purpose, we refer to IRG as *PIRD*(Page-level Inter-Reference Distance).

D. Locality Estimation and Adaptive Partitioning: A Distribution-Based Approach

Our BPAC scheme requires that the p-list and b-list share the cache space, and that the size-dependent region is dynamically adjusted in the b-list. Considering that the dual-list differentiates pages of high and low temporal locality, and that the size-independent and size-dependent regions contains clusters of high and low spatial locality respectively, we may measure the two localities, and from the measurement results, we may have hints about the time/duration for a page to stay “hot” or for the cluster to be “growing larger”.



(a) The Overall PIRD Distribution (b) Correlation between PIRD Average and Frequency

Fig. 3. Overall PIRD Distribution Analysis.

1) *Partitioning between P-list and B-list with PIRD Distribution*: BPAC keeps the hot pages in the p-list, however, most of the hot pages are only temporarily hot, and for these kind of temporary hot-spots, there must be some method to determine their lifetime in the p-list. We use the *PIRD*, which represents the interval of re-accesses to the same page to quantify the temporal locality. We obtained the “overall” PIRD distribution by putting together the PIRD’s of all hot pages’ PIRD sequences. We found that a set of PIRD values that contribute the most in the distribution is within a short range starting at 0. In Figure 3(a), the overall PIRD distribution is obtained with the entire F1 trace; as shown in the figure, a threshold of about 200 is enough to cover most of the distribution. The cause of this phenomenon is that, as the main contributor of the overall distribution, the set of hottest pages tend to have low PIRD’s: Figure 3(b) presents the correlation between the PIRD averages of the hot pages and their frequencies (hotness) with the F1 trace; as the “hotness” increases along the y axis, the corresponding PIRD average decreases.

Based on the above observations, if a small threshold is used as an estimation of the upper bound of the access intervals of the hot pages, most of PIRD sequences are expected to be covered. We refer to it as “PIRD_thd”. If a hot page is not accessed for this upper bound interval, the chance is minimal that there will still be more accesses to this page in the near future. A series of overall distributions are sampled by collecting the PIRD’s for each consecutive period (e.g., every 10k virtual time) of the workload, upon which a series of PIRD_thd’s are located. PIRD_thd is found stable within each workload of the traces we used, and this allows it to be used as the predicted lifetime of the hot pages.

2) *Partitioning between Size-Dependent and Size-Independent Region with BIRD Distribution*: Taking PIRD_thd as an analogy, we use the “BIRD_thd” that covers the most of the overall BIRD distribution (consisting of BIRD’s from all BIRD sequences) as the lifetime of the cluster staying in the size-independent region.

The *workload monitor* (Figure 1) detects the incoming page and searches in its memory for the last reference time (upon page hit) of the page or last reference time of the corresponding cluster (upon page miss). The monitor then derives the PIRD/BIRD for this access, and then inserts this PIRD/BIRD into the PIRD/BIRD distribution. When a sampling period is

TABLE II
STATISTICS OF DISK I/O TRACES

	F1	F2	Cello99-Disk3	Cello99-Disk8
Requests(10^6)	1	0.65	0.72	1
Unique pages	113561	98239	267894	249387
Total pages	1930249	1029983	1516588	2315396

over, the workload monitor will derive both PIRD_thd and BIRD_thd as the lifetimes for hot pages and hot clusters, respectively, for the next period; and it also will flush the distributions and start sampling PIRD and BIRD again.

E. Replacement Policy: Differential Treatments

Due to different spatial locality features, various access patterns require different treatment:

Sequential patterns: To avoid early evictions, sequential pattern’s spatial locality must be fully exploited (the cluster contains an entire block), before it can be evicted to the FTL. *Looping patterns*: Within a short period (shorter than the looping period), a looping pattern can be considered to be a sequential pattern. Looping patterns are detected in BPAC, and pages of looping clusters are kept in the b-list so as to not compromise their spatial locality, and their lifetime is determined by BIRD_thd. *Random patterns*: These clusters are neither sequential nor looping. Due to their low predictability of BIRD sequence, their lifetime is determined by BIRD_thd.

The *victim selector* (Figure 1) selects the cluster to be evicted in the order of “full” sequential cluster followed by the largest cluster in the size-dependent region (containing the looping or random clusters that run out of *lifetime*).

V. EVALUATION

A. Evaluation Methodology and Experiment Configuration

To verify the effectiveness of BPAC, we have conducted trace-driven simulations and compared BPAC with BPLRU, FAB, and CLC. We modified the sim-cache module of the simplescalar tool set [16] to support BPAC, BPLRU, FAB, and CLC. Three well-known real-world disk I/O traces are used as summarized in Table II. Where, *Financial 1* and *Financial 2* [17] are obtained from OLTP applications running at two large financial institutions; the *Cello99* [18] trace pool is collected from the “Cello” server that runs HP-UX 10.20. Because the entire *Cello99* is huge, we randomly use one day traces (07/27/99) of two disks (Disk 3 and Disk 8). Since a typical page size in most SSD’s is 4KB, we convert the LBA’s of the entries in the original traces to 4KB page LBA’s. In our experimental tests, each SSD block consists of 64 pages by default and the cache size ranges from 8M to 128M bytes.

B. Experimental Results and Analysis

In this section we compare BPAC with BPLRU, FAB, and CLC. For CLC, the value of the α is set to 0.1, which is its default configuration in the author’s paper [4]. BPAC adopts the mechanism of *BIRD_thd* to adaptively tune its own α value, and in our simulation tests, the sampling period of both PIRD and BIRD distributions is 10k virtual time.

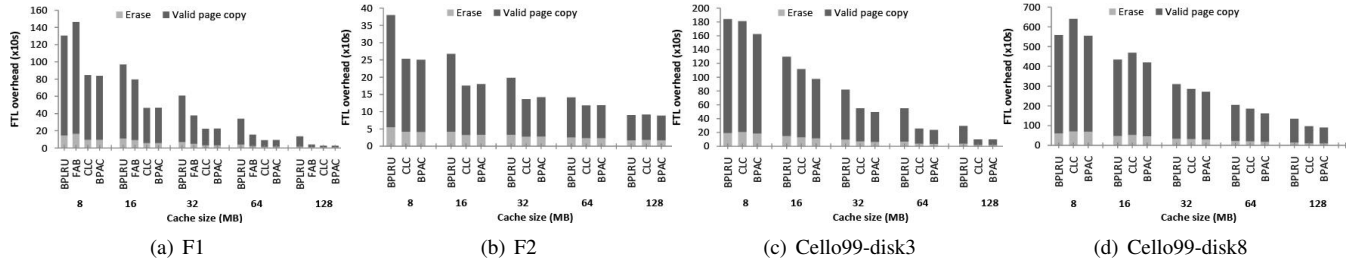


Fig. 9. Overhead on BAST FTL.

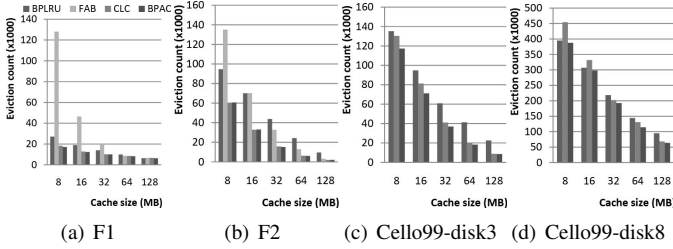


Fig. 4. Performance comparison: eviction counts.

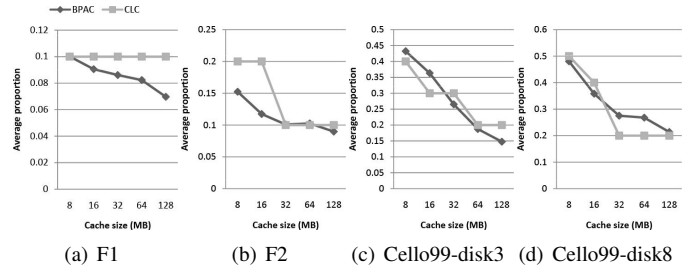


Fig. 8. The proportion of size-independent region: α .

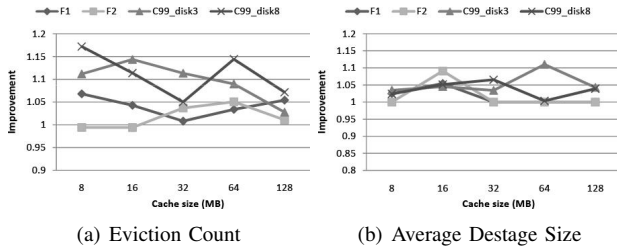


Fig. 5. Performance improvement of BPAC over CLC: eviction counts and average destage size.

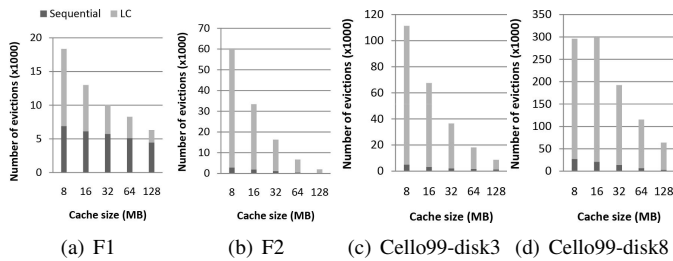


Fig. 6. Comparison between the numbers of evictions contributed by Largest Cluster(LC) policy and the “sequential” policy.

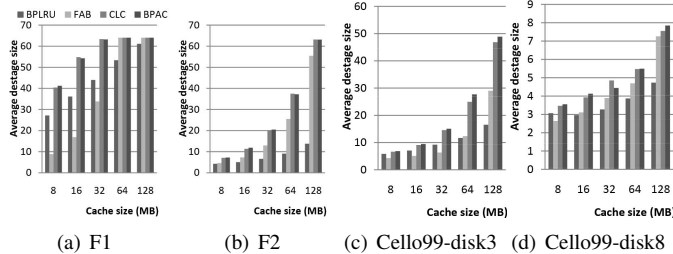


Fig. 7. Average destage size.

1) *Eviction Counts*: Our first experiment is to measure the eviction count under different cache schemes. Here the eviction count is the number of clusters evicted/destaged to the FTL. The results are shown in Figure 4. The results demonstrate that our BPAC effectively reduce the eviction count. Under F1 and F2 traces, it outperforms BPLRU and FAB and achieves comparable performance of CLC. Under the Cello99 traces, BPAC outperforms all other cache schemes (note: since FAB produces much higher eviction counts than other schemes, we exclude it in Figure 4(c) and (d)). For example, BPAC’s eviction count is 48% of BPLRU’s using the Cello99-disk3 trace and 64 MB cache. For a clear view of the improvement BPAC has over CLC, the normalized eviction counts (results of CLC over that of BPAC) are given in Figure 5(a). BPAC has higher improvement over CLC under the cello traces than it does under F1 and F2 traces, with most cache sizes. For example, with cello99_disk3, BPAC is 10% better than CLC in average; while with F2, the improvement is between 0% to 5%. We attribute the performance gain of BPAC to two aspects.

First, the effectiveness of the Largest Cluster(LC) policy on the size-dependent region: As shown in Figure 1, we use a size-dependent region to hold the clusters of low spatial locality, where the largest cluster is the candidate to be evicted. We found this policy works better than BPLRU to make more room in the cache for future new pages and existing hot pages. BPAC adopts the same policy on sequential patterns (evict sequential clusters as soon as cache space is needed), hence applying the LC policy on the random/looping pattern clusters of the size-dependent region is the main difference between the replacement policy of BPAC and BPLRU. In Figure 6, we break down the eviction counts of BPAC into two parts, one from the “LC” policy on random/looping patterns, and the other from the “sequential” policy on sequential patterns. As we can see, the main contributor for reducing the eviction

count is “LC”, which is the main source of the performance gain BPAC has over BPLRU.

Additionally, we measure the average size of the destaged clusters in Figure 7. We notice that the average destage size is inversely proportional to the eviction count, which also shows that the idea of “LC” on the size-dependent region is effective. BPAC’s improvement in the average destage size over CLC is given in Figure 5(b) (again these results are shown as a ratio of the improvement of BPAC over that of CLC).

Second, the effectiveness of BPAC’s adaptive α : The α value is the key factor that affects the eviction count, as we discussed about CLC in Section III. To learn about the effectiveness of BPAC’s mechanism of adaptively tuning α , the BIRD_thd, we average the α value of each sampling period, and compare it with the manually obtained optimal α of the CLC scheme. The results are shown in Figure 8. Due to our adaptive tuning, BPAC approaches optimal α value automatically.

For certain workloads, as the cache size increases, we expect that the proportion of the size-independent region (in which the clusters are growing larger) should decrease. In Figure 8, as the cache size increases, the decreasing trend of α in both CLC and BPAC is observed. In addition, different α ’s should be applied to different workloads to achieve better performance. For example, the largest α ’s of F1, F2, Cello99-disk3, and Cello99-disk8, are 0.1, 0.2, 0.4 and 0.5, respectively. CLC’s static α parameter is a prime hindrance in its design as manually tuning is not practical or even feasible in most cases under changing workloads.

2) *Overhead on the FTL:* We measure the overhead as the time the FTL spends on merge operations due to the shortage of log blocks. We focus on BAST FTL since FAST FTL shows similar trends. Typically a merge operation in BAST involves three steps: *read valid pages* (from the flash to page registers), *copy valid pages* (or program pages into the flash) and *erase dirty blocks*. Since the time to *read valid pages* is trivial compared to copy and erase, we do not report it in the figures. In the simulation, the number of log blocks is set to 50 and other key parameters are taken from Table I. As we discussed above, the average destage size is inversely proportional to the eviction count. Thus, the impact of the eviction count is clear: not only do smaller evictions and larger destages result in reduced numbers of merge operations, but each merge operation on average has lower overhead due to fewer clean page copy events. The results in Figure 9 show lower overhead in the FTL using BPAC. For example, the overhead in the FTL using BPAC is 40% of the overhead using BPLRU with Cello99-disk3 trace and 64 MB cache.

VI. CONCLUSION

In this paper we present BPAC, an adaptive flash-aware write cache that minimizes evictions by exploiting both spatial and temporal locality. According to temporal locality, hot pages are absorbed in the p-list, and blocks (clusters of pages) are cached in size-independent and size-dependent regions in the b-list according to their spatial locality. Simulation results show that compared to existing popular flash-aware schemes,

BPAC reduces the number of evictions and increases the size of destages which in turn reduces the overhead on the FTL, and thus improves the overall performance.

ACKNOWLEDGEMENT

This research is sponsored in part by National Science Foundation grants CCF-0937799 and CNS-0720617. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

- [1] L. Mason. (July 2009) Rethinking ssds. [Online]. Available: <http://www.denali.com/wordpress/index.php/dmr/2009/07/23/rethinking-ssds>
- [2] H. Kim and S. Ahn, “Bplru: A buffer management scheme for improving random writes in flash storage abstract,” in *FAST ’08: Proceedings of 6th USENIX Conference on File and Storage Technologies*, 2008.
- [3] H. Jo, J.-U. Kang, S.-Y. Park, J.-S. Kim, and J. Lee, “Fab: flash-aware buffer management policy for portable media players,” *IEEE Transactions on Consumer Electronics*, vol. 52, no. 2, pp. 485–493, 2006.
- [4] S. Kang, S. Park, H. Jung, H. Shim, and J. Cha, “Performance trade-offs in using nvram write buffer for flash memory-based storage devices,” *IEEE Transactions on Computers*, vol. 58, no. 6, pp. 744–758, 2009.
- [5] OCZ Technology. (June 2009) Press release: Ocz technology launches z-series power supplies and debuts new high-end solutions at this year’s computex. [Online]. Available: <http://www.ocztechnology.com/aboutocz/press/2009/341>
- [6] M. Rosenblum and J. K. Ousterhout, “The design and implementation of a log-structured file system,” *ACM Trans. Comput. Syst.*, vol. 10, no. 1, pp. 26–52, 1992.
- [7] J. Kim, J. M. Kim, S. Noh, S. L. Min, and Y. Cho, “A space-efficient flash translation layer for compactflash systems,” *IEEE Transactions on Consumer Electronics*, vol. 48, no. 2, pp. 366–375, 2002.
- [8] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S.-W. Park, and H.-J. Song, “Fast: An ftl scheme with fully associative sector translations,” in *UKC 2005*, August 2005.
- [9] J. U. Kang, H. Jo, J. S. Kim, and J. Lee, “A superblock-based flash translation layer for nand flash memory,” in *International Conference on Embedded Software*, 2006.
- [10] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li, “A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement,” in *HPCA-16: The 16th IEEE International Symposium on High-Performance Computer Architecture*. IEEE, Jan 2010, pp. 141–153.
- [11] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, “Design tradeoffs for ssd performance,” in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, Boston, Massachusetts, USA, 2008.
- [12] S. yeong Park, D. Jung, J. uk Kang, J. soo Kim, J. Lee, S. yeong Park, D. Jung, J. uk Kang, J. soo Kim, and J. Lee, “Cflru: a replacement algorithm for flash memory,” in *In CASES 06: Proceedings of the 2006 international conference on Compilers, architecture*. ACM Press. (1, 2006, pp. 234–241.
- [13] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, “Extending ssd lifetimes with disk-based write caches,” in *FAST’10: the 8th USENIX Conference on File and Storage Technologies*. USENIX, Feb 2010.
- [14] (2007) Umass trace repository. [Online]. Available: <http://traces.cs.umass.edu/index.php/Storage/Storage>
- [15] R. Karedla, J. S. Love, and B. G. Wherry, “Caching strategies to improve disk system performance,” *IEEE Computer*, vol. 27, no. 3, pp. 38–46, March 1994.
- [16] S. LLC. (2009) The simplescalar tool set. [Online]. Available: <http://www.simplescalar.com/>
- [17] Storage Performance Council. Spc trace file format specification. [Online]. Available: <http://traces.cs.umass.edu/index.php/Storage/Storage>
- [18] Hewlett-Packard Laboratories. cello99 traces. [Online]. Available: <http://tesla.hpl.hp.com/opensource/>