Boosting Random Write Performance for Enterprise Flash Storage Systems

Tao Xie
Computer Science Department
San Diego State University
San Diego, California, USA

Janak Koshia
Computer Science Department
San Diego State University
San Diego, California, USA

***Abstract*—NAND flash memory has been successfully employed in mobile devices like PDAs and laptops. With recent advances in capacity, bandwidth, and durability, NAND flash memory based Solid State Disk (SSD) is starting to replace hard disk drive (HDD) in desktop systems. Integrating SSD into enterprise storage systems, however, is much more challenging. One of the major challenges is that server applications normally demand an exceptional random I/O performance, whereas current SSD performs poorly in random writes. To fundamentally boost random write performance, in this paper we propose a new write cache management scheme called EPO (*e*lement-level *p*arallel *o*ptimization), which reorders write requests so that element-level parallelism within SSD can be effectively exploited. We evaluate EPO using a validated disk simulator with realistic server-class traces. Experimental results show that EPO noticeably outperforms traditional LRU algorithm and a state-of-the-art flash write buffer management scheme BPLRU (*b*lock *p*adding *l*east *r*ecently *u*sed).**

Keywords - Flash SSD; cache management; random write; storage system

I. INTRODUCTION

Data-intensive server applications such as OLTP (Online Transaction Processing) [15] normally demand a high-performance and highly reliable underlying storage system. Currently, rotating based magnetic hard disk drives (HDDs) are dominant building blocks for enterprise storage systems. Although they are cost-effective and can provide huge capacity and high-throughput, they are facing several serious difficulties. First of all, while disk capacity has been increasing at a rate of about 60% per year, disk access latency has only been improving about 10% per year [38]. As a result, the performance gap between disk access latency and the rest of the computer system has been widening dramatically. Second, in order to meet the 40% annual growth target of the internal data rates (IDR), HDD manufacturers have to continuously increase RPMs (revolutions per minute) and shrink platter sizes. Constantly increasing RPMs and shrinking platter sizes, however, negatively affect drive heat dissipation, which in turn causes impaired disk reliability [14]. Third, HDDs are inherently energy-inefficient and the cost of energy is increasing at an annual rate of 20%~30%, which makes energy consumption one of the largest considerations in the TCO (Total Cost of Ownership) of a data center [29]. Consequently, NAND flash memory based solid state disk (hereafter, SSD), which does not have the drawbacks mentioned above, becomes a promising alternative to HDD. Because of its solid state design, SSD is free of mechanical movements, and thus, has enhanced reliability [3]. It also inherently consumes much less energy than rotating based HDD [1][13][20]. Besides, SSD offers much faster random access by eliminating unnecessary seek time delays and rotation latencies [7][9]. It is physically robust with high vibration-tolerance and shock-resistance [5][6]. The main concern on current SSDs is their noticeably higher prices. Fortunately, the price of flash memory in the last five years has come down around 50% per year [8]. With steep annual price declines in flash memory chips, Samsung expects SSDs to reach price parity with HDDs within the next few years [8].

NAND flash memory has been successfully employed in mobile devices like PDAs and laptops [19][24]. With recent advances in capacity, bandwidth, and durability, SSD is starting to replace HDD in desktop systems [10][21]. Integrating SSD into enterprise storage systems, however, is much more challenging [5][7][13][23][31][30][36]. One of the major challenges is that off-the-shelf SSD exhibits poor random write performance [5][7][13]. Thus, simply replacing existing HDDs with SSDs in enterprise-class storage systems could lead to serious problems [2][16][32]. One of such problems is that the poor random write performance of SSDs can largely degrade the overall performance of data-intensive applications, which could generate heavy random writes with no locality [21]. For example, Dumitru compared the overall performance of a 32 GB SanDisk SATA SSD with a 36 GB Seagate 15K RPM SAS HDD under a workload with 4K operations including both reads and writes [11]. He found that even a read-dominant workload with only 10% 4K random writes and 90% 4K random reads can make the SSD's overall performance in terms of IOPS (Input/Output Operations Per Second) 1.5 times worse than that of the HDD [11]. The culprit of SSD poor random write performance is its intrinsic operating mechanism including out-of-place

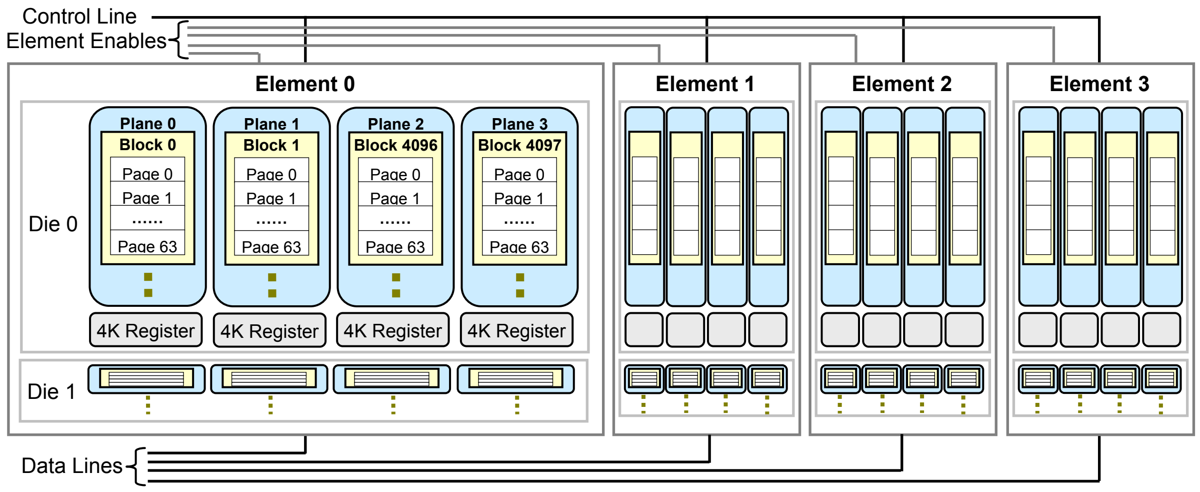


Figure 1. Internal structure of a SSD with four elements.

updating [6], time-consuming erasing and garbage collection [7], complicated FTL (flash translation layer) logic [13], and inefficient media transfer rate [10]. For example, while reading a page from the flash media into a 4 KB data register only takes 25 µs, writing a page from data register to the flash cell needs 200 µs [1]. Even worse, erasure can only be operated at block granularity and it takes 1.5 ms to erase a block [1]. Therefore, new techniques that can fundamentally boost SSD random write performance are greatly needed in order to merge SSDs into enterprise storage systems.

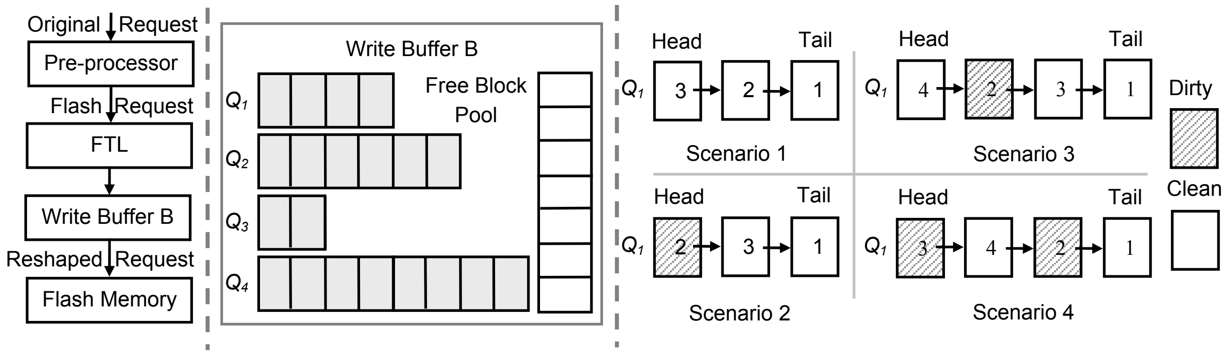
A number of investigations on SSD random write problem have been reported recently in the literature [13][21][28]. These state-of-the-art research studies can be categorized into three groups: *adding non-volatile RAM (NVRAM) buffer* [28], *enhanced FTL engine developing* [15], and *write requests buffering and reordering* [21]. Essentially, schemes in the last group employ a hardware (i.e., RAM buffer) and software (i.e., buffer management scheme) combined approach to boosting SSD random write performance. Compared with techniques in the first two groups, the last group's schemes have some advantages. First, there is no need to modify existing FTL layer, and thus, they can be easily integrated to current SSDs. Second, the RAM buffer that they required normally is less than 128 MB [21], whereas the size of NVRAM buffer demanded by techniques in the first group is in the GB scale [28]. Hence, in this research we adopt the methodology used in the last group.

However, our EPO (element-level parallel optimization) scheme takes a different approach to solving SSD random write performance issue. Unlike BPLRU [21] whose objective is to reduce the number of flushes of dirty pages from the buffer into flash memory, EPO utilizes SSD element-level concurrency by dispatching multiple buffered write requests to different elements at one time so that the requests can be processed in parallel. When page replacement occurs, all requests in one batch will be flushed to an array of elements. In this manner the re-ordered write requests can be served by multiple elements concurrently. Extensive simulations using the Microsoft SSD model [1] inside the DiskSim 4.0 simulator [35] and real-world server application traces demonstrate that EPO outperforms existing write buffer management scheme BPLRU and traditional LRU algorithm.

The rest of this paper is organized as follows. Section II briefly introduces the basics of flash memory followed by related work and the motivation of this research. The design and implementation details of the EPO scheme are presented in Section III, which is followed by the experimental results discussed in Section IV. Section V concludes this research work and points out the future research.

II. Related Work and Motivation

NAND flash memory based SSD is a semiconductor storage module, which is made of arrays of NAND flash memory elements (also called packages) [1][4]. Fig. 1 shows the internal structure of a Samsung 4 GB flash element and an array organization with four such elements [1]. Normally, an SSD has one or multiple identical elements, which can work in parallel [1]. One possible way to organize an array of elements within an SSD, as shown in Fig. 1, is that each element has separate data path to the controller but all elements share one control bus [1]. This element array structure can increase bandwidth as it supports concurrent operations that span multiple elements. Further, each element can have multiple dies (also called chips) that share one serial I/O bus and common control signals [3]. For the Samsung 4 GB flash element (Fig. 1), each die contains four planes with each having 2,048 blocks and one 4 KB data register as an I/O buffer. Each block has 64 4-KB pages. While reads and writes are page-oriented, erasure can be conducted only at block granularity [10]. A block must be erased before being



(a) (b) (c)

Figure 2. (a) Request processing flow; (b) internal structure of *B*; (c) states of Q_1 in sequence (1, 2, 3, 2, 4, 3).

programmed (written) [6]. SSD does not allow in-place update as a write operation can only change bits from 1 to 0 [10][12]. In order to mimic a HDD and expose an array of logic blocks to the upper-level components, a critical software component called flash translation layer (FTL) is implemented in the SSD controller [6][7]. Modern FTL [17][18][25][26][34] generally accomplishes three major tasks: mapping logical blocks to physical flash pages, garbage collecting, and wear leveling. Since writes cannot be performed in place, each write of a logic page must be executed on a different physical flash page. Hence, FTL maintains a mapping table to map a logic block address (LBA) to a physical block address (PBA) [7].

While there have been a number of studies [3][19][22][37] discussing flash memory random write issue, little attempt had been made until very recently [13][21][23][28] to systematically improve SSD random write performance under desktop or server workloads. Leventhal suggested adding non-volatile RAM (NVRAM) in the form of battery-backed DRAM so that writes are committed to the NVRAM ring buffer and immediately acknowledged to the client while the data is written out to the SSDs later [28]. Although this technique allows for a tremendous improvement for synchronous writes, its drawbacks are also obvious. It requires very expensive NVRAM, whose maximum size (normally 2 ~ 4 GB) is still so small that server-class workloads can fill the entire ring buffer before it can be flushed to SSD [30]. Gupta *et al.* [13] proposed DFTL (Demand-based Flash Translation Layer), which selectively caches page-level address mappings using existing SRAM cache on SSDs. Its advantage is that it does not require any extra hardware. Nevertheless, the main downside of DFTL is that it requires modifying existing FTL layer, which would bring about a high cost for it to be integrated into current SSDs. Kim and Ahn suggested embedding a small-scale (e.g., 1 ~ 16 MB) RAM write buffer into SSDs and proposed BPLRU (block padding least recently used) [21], a flash flavour variant of the well-known LRU (least recently used) scheme, to manage the buffer.

Several recent investigations [1][7][10] on SSD internal characteristics and system-level organization analyzed the multi-level concurrency presented in SSDs, which inspires us to develop a parallelism-driven software/hardware combined scheme to improve random write performance. The EPO scheme judiciously reshapes a random write access pattern to a parallelism-aware batch-based write stream so that the element-level concurrency can be mostly utilized to improve SSD random write performance. EPO opens a new avenue to boost SSD random write performance as it is orthogonal to existing approaches.

III. The EPO Scheme

We now present our EPO scheme that exploits the element-level parallelism in SSD to boost random write performance. The basic idea of EPO is illustrated using an example followed by a detailed description.

A. An Illustrative Example

Fig. 2a illustrates the request processing flow of EPO. Similar to BPLRU [21], EPO only processes write requests. For read requests, it simply forwards them to the FTL. An *original request* issued by the host is first processed by a *pre-processor*, which transforms it into a *flash request* by replacing its size to the closest times of the size of a flash page (e.g., 4 KB). As a result, the size of a flash request is either equal to one flash page or multiple flash pages (see Fig. 1). Each flash request then enters the FTL layer where its logical address is mapped to a physical address. Next, every request goes to the write buffer B, which is a RAM buffer embedded in an SSD. To prevent data loss in the event of a power failure, we assume that the RAM buffer B is protected via an onboard battery. Finally, EPO manages flash requests in the write buffer B and outputs *reshaped requests* to flash memory.

The basic unit in the write buffer B is a block whose size is equal to the size of a flash memory page (e.g., 4 KB in our experiments). Within B, EPO maintains a *free block pool* and multiple queues (Fig. 2b). Each queue is a linked-list that contains all requests visiting one particular element. The number of queues in B is equal to the number of elements in the underlying SSD.

Assume that there are only four elements in an SSD, Fig. 2b demonstrates how EPO manages the free block pool and the four queues with each queue corresponding to one element. When a new request arrives in B, EPO acquires one empty block from the free block pool to accommodate the new request, and then, places it into its corresponding queue based on its physical address. After more requests enter B, the free block pool shrinks until it is empty. At this moment, victims need to be chosen when a new request comes. Every time when free space is needed for a new request while the free block pool is empty, EPO selects multiple victims each from one queue and flushes them to SSD. One of the freed blocks will be used for the new request and all others shall be reclaimed into free block pool. We will illustrate how EPO selects the victims next.

Assume that all six requests shown in Fig. 2c are single-page requests and they all target on element one. Also, suppose that at the beginning three requests with distinct physical addresses (i.e., 1, 2, and 3) sequentially arrive in B. EPO inserts the three requests into Q_1 one by one. Consequently, request with address 1 is at the tail and the request with address 3 stays in the head position (see Scenario 1 in Fig. 2c). Assume that the physical address of the fourth request is also 2. In other words, the fourth request would overwrite the second request, which has been located in Q_1. Therefore, EPO updates the address 2 block with the content in the fourth request and then moves it to the head of Q_1 (see Scenario 2 in Fig. 2c). Now block 2 becomes a dirty block, whereas block 1 and 3 are all clean. After a while, the fifth request with physical address 4 comes. EPO simply inserts it into the head of Q_1 (see Scenario 3 in Fig. 2c). The last request again accesses the block address 3 as the third request does. Thus, EPO moves block 3 into the head of Q_1 (see Scenario 4 in Fig. 2c). If at this time a victim is needed, EPO will evict first request with block address 1, i.e., the tail of the queue Q_1. Apparently, EPO also exploits the temporal locality.

B. *Algorithm Description*

Fig. 3 outlines the algorithm of the EPO scheme. Note that the input P of the EPO algorithm is the output of the FTL layer (Fig. 2a), which translates each flash-style write request into a physical request. A flash-style request is a request whose size is equal to either one flash page or multiple flash pages. The size of a flash page is normally 2 KB or 4 KB for modern SSDs. Another input B is an empty buffer managed by EPO. The write buffer B is implemented as a RAM cache inside an SSD and its size is configured to be in the range 4 — 32 MB in our simulations. The output of the EPO scheme is a reshaped single-page write request set R, which is then fed into the flash SSD (Fig. 2a). More importantly, the request stream in R can mostly exploit the element-level concurrency in the flash SSD.

Now we describe each step in Fig. 3 in detail. First, EPO creates e empty queues from Q_1 to Q_e in B, where e is the number of elements in SSD. Each queue will be used to maintain requests that target on a particular element. Next, EPO processes each request in P within the outmost loop (Step 2 ~ 29) in the same manner. EPO first judges whether a request in P is a multiple-page request or a single-page request (Step 5). If the request is a multiple-page request, EPO breaks it down into multiple single-page requests (Step 6). Consequently, only single-page requests will be maintained in each queue, which in turn increases the element-level concurrency later. For each newly arrived single-page request, EPO searches its physical address in its corresponding queue, which is a linked-list data structure. If the same address is found in an existing request in the queue, the existing request will be overwritten by the new single-page request. Further, the new request will be moved to the head of the queue so that the temporal locality can be utilized (Step 15). If the same physical address in the queue is not found,

```
Input: P, a pre-processed write request set; B, a write buffer
managed by the EPO scheme
Output: R, a re-shaped write request set that is aware of the
element-level parallelism in SSD
1.  Clear B and R; k = 1; e = number of elements in SSD;
    create e queues from $Q_1$ to $Q_e$ in B
2.  for each request $r_k \in P$ do
3.    j = number of pages requested by $r_k$
4.    Create a temporary array T with j cells
5.    if j > 1 /* the request $r_k$ is a multiple-page request */
6.      Divide the request $r_k$ into j single-page write requests
        and store them in T sequentially
7.    else
8.      Store $r_k$ in T
9.    end if
10.   h = 1
11.   for each single-page write request $t_h \in T$ do
12.     i = the element number that $t_h$ targets on and $1 \le i \le e$
13.     Search $t_h$ in the corresponding queue $Q_i$ in B
14.     if the page requested by $t_h$ is found in $Q_i$
15.       Replace it with $t_h$ and move $t_h$ to the head of $Q_i$
16.     else
17.       if there is no free space in B to accommodate $t_h$
18.         for each queue from $Q_1$ to $Q_e$ in B
19.           Evict the request at the tail to R
20.           Change its arrival time to the arrival time of $t_h$
21.         end for
22.       end if
23.       Insert $t_h$ at the head of $Q_i$
24.       The free block pool is increased by e - 1 blocks
24.     end if
25.     h = h + 1
26.   end for
27.   Delete the temporary array T
28.   k = k + 1
29. end for
```

Figure 3. Algorithm of the EPO scheme.

EPO needs to acquire a free block from the *free block pool* to accommodate the new request (Fig. 2b). When the *free block pool* is not empty, EPO inserts the new quest at the head of the queue. Otherwise, EPO starts to select *e* victims each from one queue (Step 18 ~ 21). Each request at the tail of a queue will be chosen as a victim and its arrival time is updated to the arrival time of the new request (Step 19 ~ 20). All victims are then inserted into *R*, which contains requests to the SSD. While the original request set *P* exhibits a random access pattern, the reshaped request set *R* becomes a batch-based request stream. Requests that are evicted from *B* at the same time are bundled together to form a batch. Requests in one batch go to different elements of an SSD and they share the same arrival time. Thus, all elements can serve requests in one batch in parallel.

IV. Performance Evaluation

In this section, we present our experimental results for a variety of configurations including write buffer size, number of elements, and flash page size. Mean response time will be the primary performance metric in this study. We also measure throughput for all of the four algorithms. Three real system traces Financial1 [33], Financial2 [33], and TPC-C [27] are used in this simulation study to evaluate the performance of EPO as well as NoCache, LRU, and BPLRU. In Section IV.A, the experimental settings for the simulations are described. In addition, we investigate the impact of write buffer size in Section IV.B and the impact of the flash page size in Section IV.C. Finally, we study the scalability of the four schemes in Section IV.D.

A. Experimental Setup

All simulation experiments are conducted in three stages sequentially: *pre-processing*, *reshaping*, and *feeding*. In the pre-processing stage, the pre-processor (Fig. 2a) performs the following tasks. First, it filters out all read requests from an original trace. Second, it truncates the entire write request set so that only the first several millions of write requests will be used later. Third, for large logical space trace like TPC-C, it evenly shrinks the trace's logical address space so that each write request's logical address can be mapped to a physical address within the scope of an SSD configuration. Finally, it changes each write request's size to its closet multiples of 4 KB pages. The output of the pre-processor is called *flash request trace* (Fig. 2a) because it is a trace suitable for flash disk storage systems. In the reshaping stage, we implemented and run the four buffer management schemes NoCache, LRU, BPLRU, and EPO on a Dell PowerEdge 1900 server with two Quad Core Intel® Xeon® E5310 1.60 GHz processors and 8GB FB-DIMM memory. After the FTL layer, all requests in a flash request trace are then buffered in the writer buffer B (Fig. 2a) and are managed by a particular scheme like EPO. The output of the write buffer B is a *reshaped request set*, which contains all requests evicted from the buffer according to the victim selection policy of a buffer management scheme. In the final feeding stage, requests in the reshaped request set are fed into the Microsoft SSD model [1], which was derived from the generic rotating disk module for DiskSim 4.0 [35]. DiskSim emulates a hierarchy of storage components including buses, controllers, and disks [35]. It is a well-known and validated disk system simulator [35].

We evaluate the four buffer management schemes by running simulations over three real system traces: Financial1 [33], Financial2 [33], and TPC-C [27], which have been widely used in the literature. Financial1 and Financial2 were collected from requests to OLTP applications at two large financial institutions. While the Finanical1 trace is write-dominant (more than 60% requests are writes), Financial2 trace is read-dominant (more than 80% requests are reads). TPC-C is an I/O trace collected on a storage system connected to a Microsoft SQL Server via storage area network [27]. The mean write request size of TPC-C is larger than 10 KB. Financial1 trace exhibits obvious temporal locality and spatial locality, whereas TPC-C trace manifests itself as a completely random access pattern. We selected the three traces so that the EPO scheme can be evaluated under different degrees of access randomness. Since the simulation times in our experiments are much shorter compared with the time spans of the traces, we truncate each trace such that only the first 2, 0.65, and 2 million write requests are included for Finanaical1, Financial2, and TPC-C, respectively. The main simulation parameters are shown in Table I.

Table I. Simulation Parameters

Parameter	Value (Fixed) – (Varied)
Write buffer capacity (MB)	(8) – (4, 8, 16, 32)
Number of elements	(48) – (16, 32, 48, 64)
Page size (KB)	(4) – (1, 2, 4)
Flash block size (page)	(64)
Element capacity (GB)	(4)
Flash SSD capacity (GB)	(192) – (64, 128, 192, 256)
Block erase latency (μs)	(1500)
Page read latency (μs)	(25)
Page write latency (μs)	(200)
Chip transfer latency per byte (μs)	(0.025)
Number of planes in an element	(8)

The BPLRU algorithm [21] is recognized as a state-of-the-art flash write buffer management scheme. On the other hand, the LRU algorithm is a widely employed

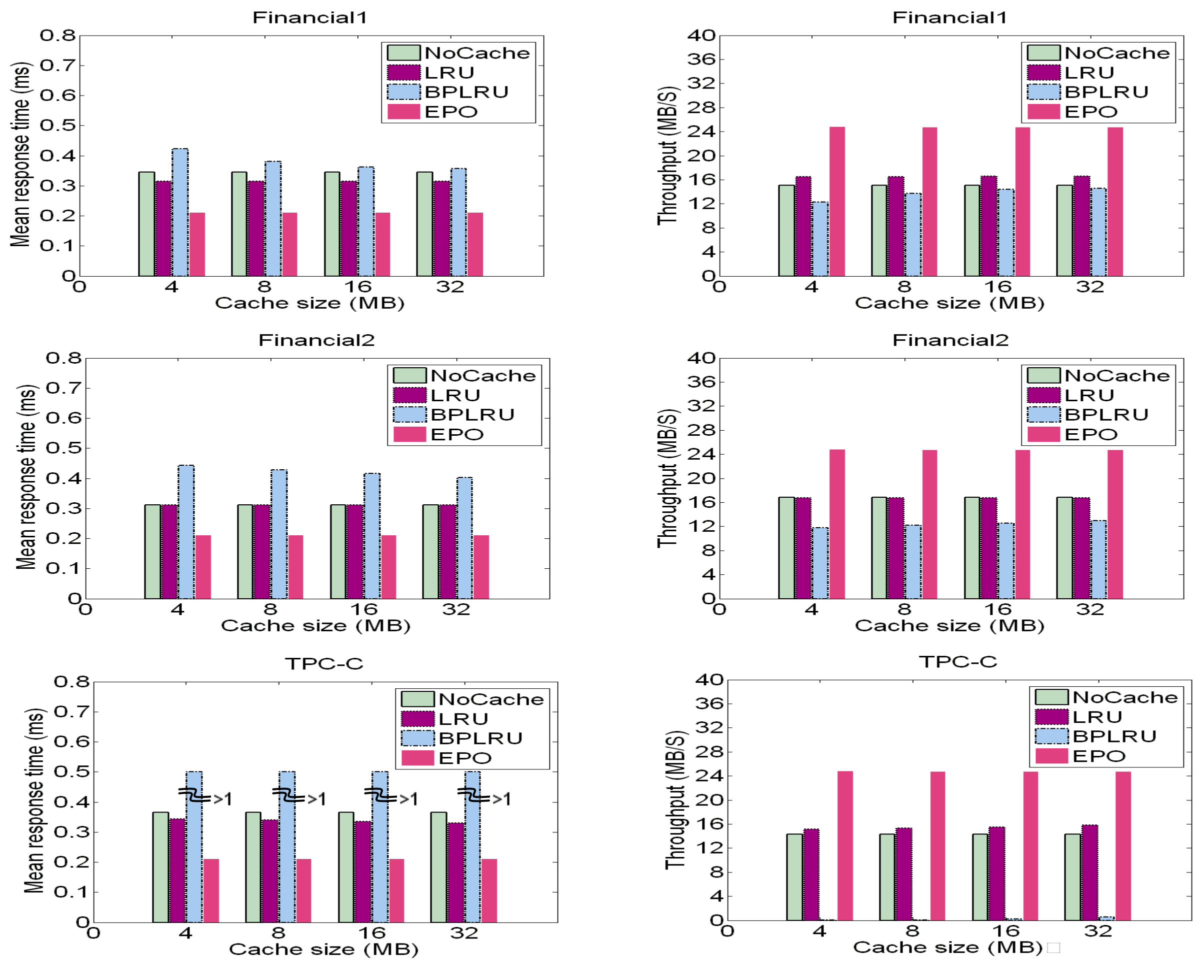


Figure 4. Performance impact of write buffer size on the four schemes.

cache management policy in real systems. We also tested the situation where no write buffer is used at all. We call this scheme NoCache, which can demonstrate the performance improvement due to adding a write buffer into SSD. In order to comprehensively evaluate the EPO scheme, we compare it with the three algorithms in this section. A brief introduction of the three algorithms is presented below.

(1) NoCache: NoCache forwards each arrival write request directly to SSD without taking any actions. It serves as a baseline algorithm.

(2) LRU (Least Recently Used): LRU maintains the "age" of each buffered write request and evicts the least recently used write request first.

(3) BPLRU (Block Padding Least Recently Used): When free space is needed to accommodate new writes, BPLRU selects the least recently used flash block rather than a sector as a victim and flushes all sectors in the victim block to flash device [21]. To reduce the buffer flashing cost, BPLRU employs a page padding technique for a victim block. Before a victim block is kicked out of the buffer, BPLRU reads all pages that are not presented in the victim block from data block. And then it writes all sectors in the victim block sequentially onto a log block. In this way, an expensive full merge can be replaced by an efficient switch merge. Detailed information about page padding and the three types of merge can be found in [22].

B. Overall Comparisons

The goal of this experiment is to compare EPO against two well-known cache management algorithms LRU and BPLRU, and to understand the impact of write buffer size on the performance of the four algorithms including NoCache. We tested write buffer size from 4 MB to 32 MB with 48 elements.

We observe from Fig. 4 that the mean response time of all four schemes does not noticeably change when the size of the write buffer increases from 4 MB to 32 MB. This is because the write buffer is still very small considering the large volume of requests from the three server-class workloads. Consequently, the entire write buffer even in its maximal size 32 MB is quickly filled

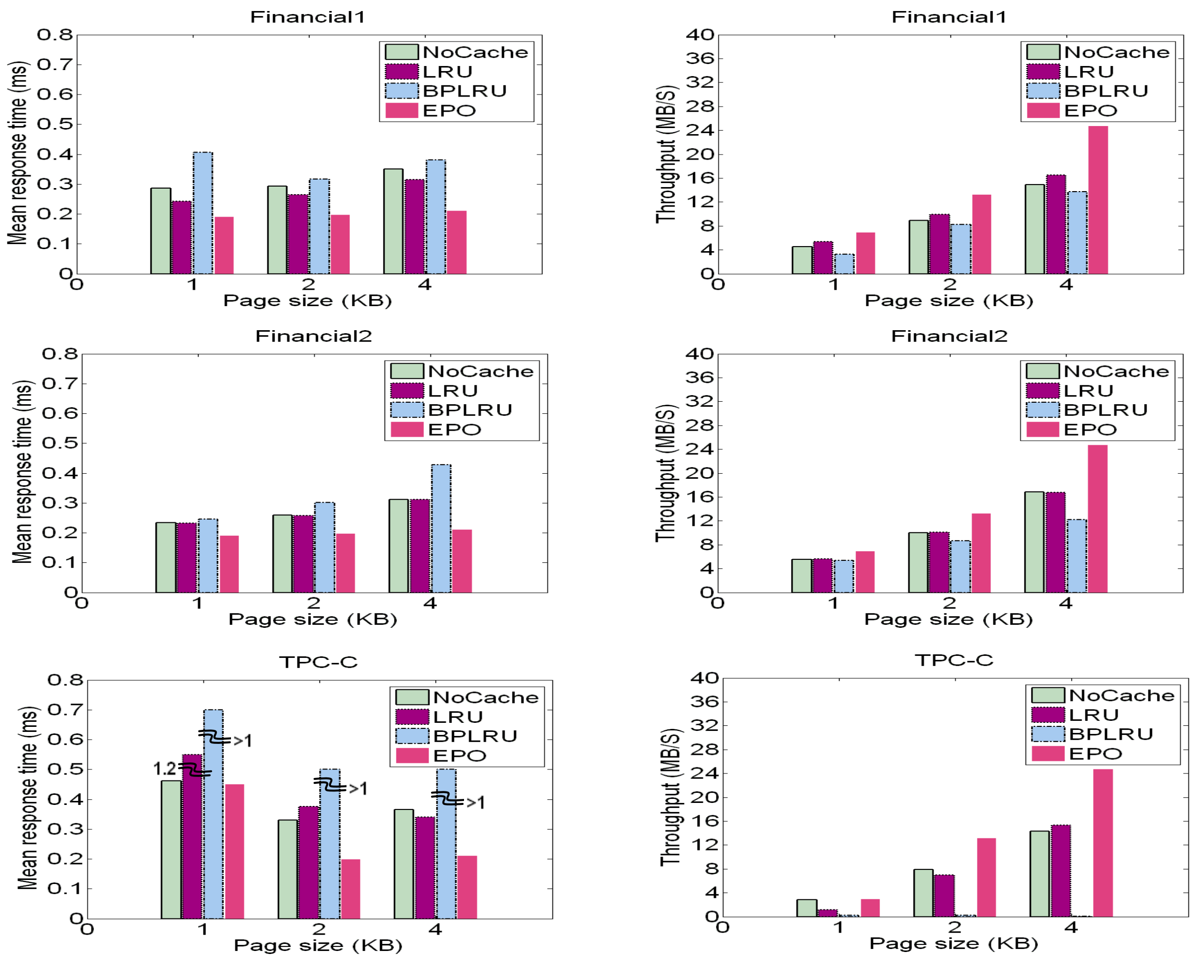


Figure 5. Performance impact of flash page size on the four schemes.

out by arrival requests, and thus, increasing write buffer size does not result in an apparent performance improvement. Still, EPO always outperforms the three existing schemes in all cases for it exploits the element-level concurrency. In Financial1 scenario, compared with NoCache, LRU, and BPLRU, EPO on average reduces mean response time by 38.9%, 33.1% and 44.6%, respectively, while on average increases throughput by 63.8%, 49.5%, and 79.8%, respectively (Fig. 4). Note that in all scenarios in Fig. 4, the performance of NoCache keeps constant as write buffer size has no impact on it. Interestingly, BPLRU exhibits the worst performance in all situations shown in Fig. 4. This is because BPLRU always assumes that logically consecutive pages must also be physically located into one block, which is not true when striping is applied within SSDs. As a result, BPLRU generates a large number of unnecessary reads and writes due to page padding [21]. In Financial2 case, EPO also outperforms the other three algorithms (Fig. 4). In TPC-C workload, compared with NoCache, LRU, and BPLRU, EPO on average shrinks mean response time by 42.3%, 37.6%, and 99.9%, respectively. BPLRU, however, experiences very large mean response times in TPC-C trace, which also explains why the throughput of BPLRU is so low in TPC-C workload in Fig. 4. Compared with NoCache and LRU in TPC-C workload, EPO on average increases throughput by 73.2% and 60.3%, respectively. Since the access pattern of TPC-C is more random than that of Financial1, EPO fully exhibits its strength and its performance improvements become more noticeable. Obviously, the performance gain achieved by EPO is at the cost of adding a RAM buffer inside SSD. Considering the small size of the buffer and the substantial performance improvement, we argue that the benefits of EPO outweigh its cost.

One interesting observation from Fig. 4 is that increasing the size of write buffer can neither significantly reduce the mean response time nor increase throughput. The rationale behind is that larger buffer size has little impact on a totally random access pattern. To understand the sensitivity of EPO to other

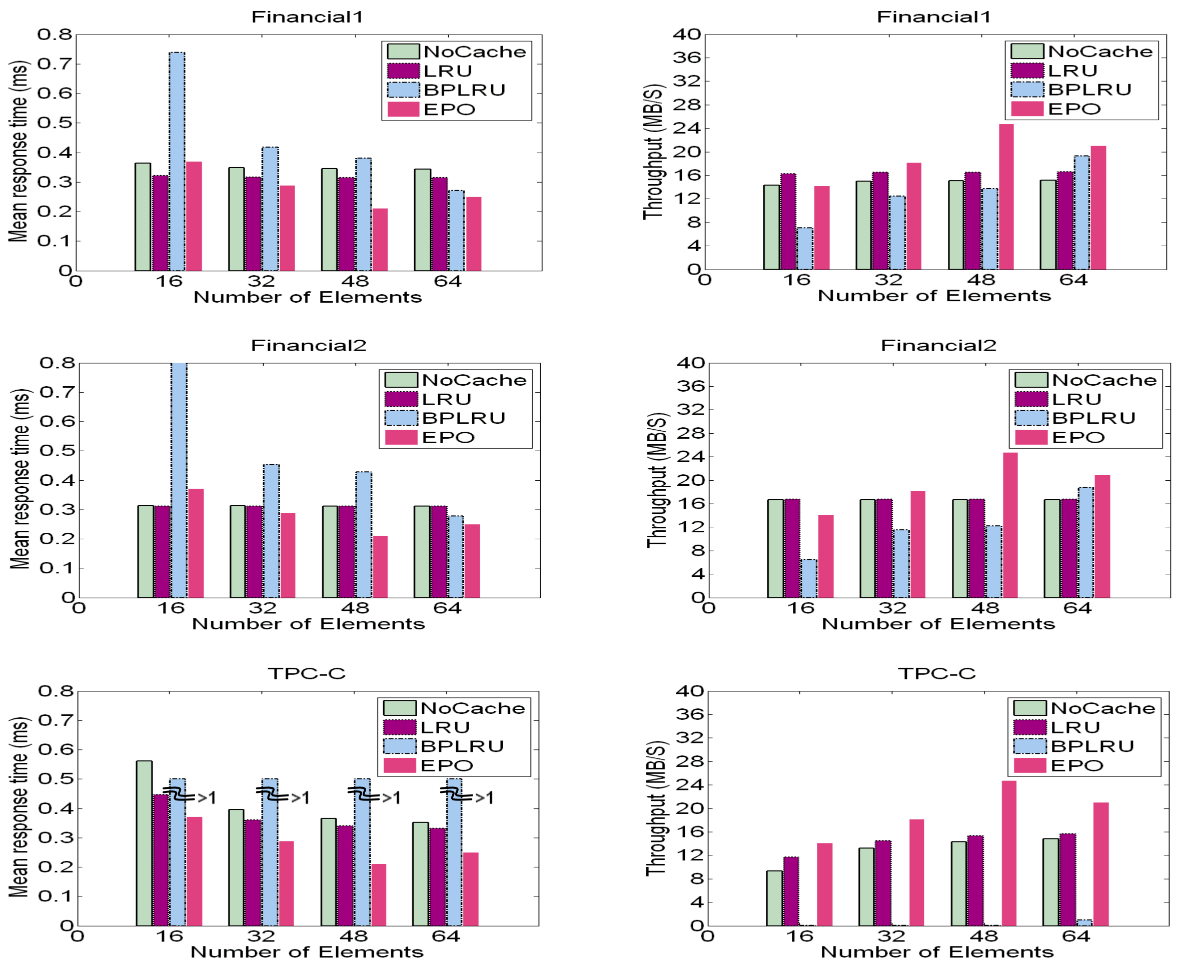


Figure 6. Scalability of the four schemes.

parameters, we also measured the performance of EPO when changing the number of elements and page size.

C. The Impact of Flash Page Size

This experiment is intended to investigate the impact of flash page size on the EPO scheme. We vary the size of a flash page from 1 KB to 4 KB. Fig. 5 plots the performance of the four algorithms as functions of the size of a flash page.

Several important observations can be drawn from Fig. 5. First of all, flash page size has a noticeable impact on the three existing algorithms. For example, under Finanical1 workload LRU increases its mean response time from 0.24 ms to 0.31 ms when flash page size varies from 1 KB to 4 KB. Meanwhile, NoCache scheme increases its mean response time by 21.3% (Fig. 5). Recall that after the pre-processing stage each write request's size is configured to its closest multiples of flash pages and each page is 4 KB (see Section IV.A). Therefore, when flash page size enlarges to 4 KB, each request needs to write multiple pages rather than a single page. Therefore, the response time of NoCache and LRU increases. The mean response time of EPO, however, only slightly changes because it always splits each multiple-page request into multiple single-page requests (Step 6 in Fig.3). Second, larger page size usually results in a higher throughput. In Financial 1 case, EPO increases the throughput by 4.9 times when flash page size changes from 1 KB to 4 KB. The reason is that larger flash page improves write efficiency and decreases the number of block erasures [1]. Lastly, TPC-C workload is so intensive that all three existing algorithms encounter large mean response times.

D. Scalability

This experiment is intended to investigate the scalability of the EPO scheme. We scaled the number of elements in an SSD from 16 to 64. The size of write buffer is set to 8 MB. Fig. 6 plots the performance of the four algorithms as functions of number of elements.

The results show that only EPO and BPLRU algorithms exhibit a good scalability in Financial1 and

Financial2 workload when the number of elements increases from 16 to 48. Specifically, in Financial1 EPO reduces its mean response time by 42.3%, whereas BPLRU improve their mean response time performance by 66.1%. Still, in terms of mean response time EPO on average outperforms NoCache, LRU, and BPLRU by 20.5%, 12%, and 38.3% in Financial1 (Fig. 6). After the 48 element case, however, none of the four algorithms show improvement in both mean response time and throughput. The reason is that the footprint of the TPC-C trace becomes relatively very small compared with the enlarged capacity of the SSD due to the increment of the number of elements. Thus, adding more elements after 48 does not help.

Overall, Fig. 6 demonstrates that the scalability of all algorithms including EPO is sensitive to the workloads. In Financial1 and Financial2 cases, increasing the number of elements does bring an apparent improvement in either mean response time or throughput (Fig. 6). After analyzing the two traces we realized that the outcome is expected because both Financial1 and Financial2 workloads have noticeable temporal locality and spatial locality. As a result, a large portion of requests concentrate on a small logical space so that newly added elements cannot receive enough requests to share the entire load. In Finanical2 case, compared with NoCache and LRU, EPO on average reduces mean response time by 10.5% and 10.2%, respectively. Compared with NoCache, LRU, and BPLRU, EPO on average improves throughput by 16.6%, 16.2%, and 59.1%, respectively. In TPC-C scenario, EPO significantly outperforms all three existing algorithms in terms of throughput. This is because EPO fully employs the element-level parallelism within an SSD.

V. Conclusions

In this paper, we address the issue of SSD random write performance in server applications. EPO (element-level parallel optimization), a new write buffer management scheme, is developed. The basic idea of EPO is to reshape write access pattern by dynamically grouping multiple buffered write requests that target on distinct elements into one batch. EPO exploits the element-level concurrency to significantly shorten mean response time and improve throughput. Although EPO also employs an extra battery-backup RAM buffer inside SSD and reshapes write access pattern, it is orthogonal to current *write requests buffering and reordering* schemes because it seeks to exploit element-level parallelism within SSD, which is a new avenue to solve the SSD random write problem. Comparing with *adding non-volatile RAM (NVRAM) buffer* and *enhanced FTL engine developing* approaches (see Section I), EPO has several desired advantages. First, its hardware cost is low because of the limited size of RAM buffer used. Second, it does not require any change in the FTL layer, and thus, is easy to be integrated into modern SSDs. Lastly, its low time complexity implies its potential to be implemented in real applications. Further, the performance of EPO plus three existing schemes including LRU and BPLRU [21] has been thoroughly evaluated using a validated simulator Microsoft SSD model [1] with DiskSim 4.0 [35] and three widely used enterprise-level traces. Experimental results demonstrate that EPO consistently outperforms a state-of-the-art write buffer management scheme BPLRU. It also performs better than the traditional LRU algorithm. When SSDs are in their default configurations, compared with NoCache and LRU, EPO achieves improvement in mean response time by up to 63.8% and 49.5%, respectively. Note that the FTL layer will not affect the performance of EPO because it is underneath the FTL layer.

All our experiments employ the Microsoft SSD model [1], which only has a generic FTL layer to accomplish tasks like logic address mapping. To understand the combined effects of EPO and some representative FTL schemes such as FAST [25], DFTL [13], and HAT [17] on the performance of enterprise flash SSDs, we are extending the SSD model so that these specific FTL algorithms can be integrated. We will then re-examine the impacts of EPO underneath various modern FTLs. Also, the Microsoft SSD model in its current format does not provide any cache management function. Another future direction of this study is to implement new buffer management schemes inside the SSD model so that their performance can be evaluated and compared. Finally, we are going to develop SSD disk array level cache management schemes to not only improve random write performance but also prolong SSD life-time by evenly distributing writes in an SSD array.

Acknowledgment

We thank DiskSim developers [35] who provided an efficient, accurate, highly-configurable disk system simulator for storage research community. Also, we would like to thank the authors of [1] from Microsoft Research who developed the SSD model for DiskSim 4.0. Finally, we are grateful to the anonymous reviewers whose insightful comments greatly improve the quality of this research work. This work was sponsored by the US National Science Foundation under grants CNS-0845105, CNS-0834466, and CCF-0702781.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," *Proc. USENIX Annual Technical Conference*, pp. 57-70, 2008.

[2] M. Balakrishnan1, A. Kadav, V. Prabhakaran, and D. Malkhi, "Differential RAID: Rethinking RAID for SSD Reliability," *Proc. 5th ACM European Conf. Computer Systems (EuroSys)*, Paris, France, April 13-16, 2010.

[3] A. Birrell, M. Isard, C. Thacker, and T. Wobber, "A design for high-performance flash disks," *ACM SIGOPS Operating Systems Review*, Vol. 41, Issue 2, pp. 88-93, April 2007.

[4] S. Boboila and P. Desnoyers, "Write Endurance in Flash Drives: Measurements and Analysis," *Proc. 8th USENIX Conference on File and Storage Technologies (FAST)*, 2010.

[5] K. Cash, "Flash Solid State Disks - Inferior Technology or Closet Super-star?" *BiTMICRO Networks*, http://www.storagesearch.com/bitmicro-art1.html. 2007.

[6] L.P. Chang and T.W. Kuo, "Efficient management for large-scale flash-memory storage systems with resource conservation," *ACM Transactions on Storage*, Vol. 1, No. 4, pp. 381-418, 2005.

[7] F. Chen, D.A. Koufaty, and X. Zhang, "Understanding Intrinsic Characteristics and System Implications Of Flash Memory based Solid State Drives," *Proc. the Eleventh Int'l Joint Conf. Measurement and Modeling of Computer Systems (SIGMETRICS)*, pp. 181-192, 2009.

[8] B. Crothers, "Samsung: Solid state will match hard-drive price," *CNET.NEWS*, http://news.cnet.com/8301-13924_3-10196422-64.html, March 15, 2009.

[9] P. Desnoyers, "Empirical Evaluation of NAND Flash Memory Performance," *Workshop on Hot Topics in Storage and File Systems (HotStorage'09)*, Big Sky, Montana, Oct. 11th 2009.

[10] C. Dirik and B. Jacob, "The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization," *Proc. 36th Int'l Symp. on Computer Architecture (ISCA)*, pp. 279-289, 2009.

[11] D. Dumitru, "Understanding Flash SSD Performance," http://managedflash.com/news/papers/easycoflashperformance-art.pdf, August 2007.

[12] L.M. Grupp, A.M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P.H. Siegel, and J.K. Wolf, "Characterizing flash memory: anomalies, observations, and applications," *Proc. 42nd Annual IEEE/ACM Int'l Symp. Microarchitecture (Micro)*, pp. 24-33, 2009.

[13] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," *Proc. Fourteenth Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 229-240, 2009.

[14] S. Gurumurthi, A. Sivasubramaniam, and V.K. Natarajan, "Disk Drive Roadmap from the Thermal Perspective: A Case for Dynamic Thermal Management," *Proc. 32nd Int'l Symp. Computer Architecture (ISCA)*, pp. 38-49, 2005.

[15] S. Harizopoulos, D.J. Abadi, S. Madden, and M. Stonebraker, "OLTP through the looking glass, and what we found there," *Proc. ACM Int'l Conference on Management of Data (SIGMOD)*, pp. 981-992, 2008.

[16] J. He, A. Jagatheesan, S. Gupta, J. Bennett, and A. Snavely, "DASH: a Recipe for a Flash-based Data Intensive Supercomputer," *The Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC'10)*, New Orleans, LA, November 13-19, 2010.

[17] Y. Hu, H. Jiang, D. Feng, L. Tian, S. Zhang, J. Liu, W. Tong, Y. Qin, and L. Wang, "Achieving page-mapping FTL performance at block-mapping FTL cost by hiding address translation," *Proc. IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1-12, 2010.

[18] D. Jung, J-U Kang, H. Jo, J. Kim, and J. Lee, "Superblock FTL: A Superblock-Based Flash Translation Layer with a Hybrid Address Translation Scheme," *ACM Transactions on Embedded Computing Systems*, Volume 9, Issue 4, March 2010.

[19] A. Kawaguchi, S. Nishioka, and H. Motoda, "A flash-memory based file system," *Proc. of the USENIX Technical Conference*, 1995.

[20] T. Kgil, D. Roberts, and T. Mudge, "Improving NAND Flash Based Disk Caches," *Proc. 35th Int'l Symp. on Computer Architecture (ISCA)*, pp. 327-338, 2008.

[21] H. Kim and S. Ahn, "BPLRU: a buffer management scheme for improving random writes in flash storage," *Proc 6th USENIX Conference on File and Storage Technologies (FAST)*, pp. 239-252, 2008.

[22] H. Kim, J.H. Kim, S. Choi, H. Jung, and J. Jung, "A page padding method for fragmented flash storage," *Lecture Notes in Computer Science*, 4705, pp. 164-177, 2007.

[23] I. Koltsidas, and S. D. Viglas, "Flashing Up the Storage Layer," *Proc. 34th Int'l Conf. on Very Large Data Bases (VLDB)*, Auckland, New Zealand, August 24-30, 2008.

[24] S. Lee, B. Moon, C. Park, J. Kim, and S. Kim, "A case for flash memory SSD in enterprise database applications," *Proc. Int'l Conf. on Management of Data (SIGMOD)*, pp. 1075-1086, 2008.

[25] S. Lee, D. Park, T. Chung, D. Lee, S. Park, H. Song, "A Log Buffer based Flash Translation Layer Using Fully Associative Sector Translation," *IEEE Transactions on Embedded Computing Systems*, 6(3):18, 2007.

[26] S. Lee, D. Shin, Y. Kim, and J. Kim, "LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems," *Proc. Int'l Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability (SPEED)*, 2008.

[27] S.T. Leutenegger and D. Dias, "A modeling study of the TPC-C benchmark," *Proc. ACM International Conference on Management of Data (SIGMOD)*, 22(2), pp. 22–31, June 1993.

[28] A. Leventhal, "Flash Storage Memory," *Communications of the ACM*, Vol. 51, No. 7, pp. 47–51, 2008.

[29] F. Moore, "Storage and Energy – the Heat is On," Horison Information Strategies, http://www.horison.com, July 2007.

[30] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating Enterprise Storage to SSDs: Analysis of Tradeoffs," *Proc. 4th ACM European Conf. on Computer Systems (EuroSys)*, pp. 145-158, 2009.

[31] K. Park, D-H Lee, Y. Woo, G. Lee, J-H Lee, and D-H Kim, "Reliability and performance enhancement technique for SSD array storage system using RAID mechanism," *Proc. 9th Int'l Conf. Communications and Information Technologies*, pp. 140-145, 2009.

[32] M. Polte, J. Simsa, and G. Gibson, "Enabling enterprise solid state disks performance," *Proc. Workshop on Integrating Solid-state Memory into the Storage Hierarchy*, Mar. 2009.

[33] SPC, "Storage Performance Council I/O traces," http://www.storageperformance.org/.

[34] J.Y. Shin, Z.L. Xia, N.Y Xu, R. Gao, X.F. Cai, S. Maeng, and F.H. Hsu, "FTL design exploration in reconfigurable high-performance SSD for server applications," *Proc. 23rd Int'l Conf. Supercomputing*, pp. 338-349, 2009.

[35] The DiskSim Simulation Environment (v4.0), http://www.pdl.cmu.edu/DiskSim/.

[36] D. Tsirogiannis, S. Harizopoulos, and M.A. Shah, "Query Processing Techniques for Solid State Drives," *Proc. ACM Int'l Conference on Management of Data (SIGMOD)*, pp. 59-71, 2009.

[37] G. Wu, B. Eckart, and X. He, "BPAC: An Adaptive Write Buffer Management Scheme for Flash-based Solid State Drives," *Proc. IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, May 6-7, 2010.

[38] X. Yu, "Trading capacity for performance in disk arrays," Ph.D. Dissertation, Princeton University, New Jersey, USA, 2004.