Front. Comput. Sci., 2016, 10(5): 951–964
DOI 10.1007/s11704-016-5286-z

RESEARCH ARTICLE

Prober: exploiting sequential characteristics in buffer for improving SSDs write performance

Wen ZHOU, Dan FENG, Yu HUA, Jingning LIU (✉), Fangting HUANG, Yu CHEN, Shuangwu ZHANG

Wuhan National Laboratory for Optoelectronics, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

Abstract Solid state disks (SSDs) are becoming one of the mainstream storage devices due to their salient features, such as high read performance and low power consumption. In order to obtain high write performance and extend flash lifespan, SSDs leverage an internal DRAM to buffer frequently rewritten data to reduce the number of program operations upon the flash. However, existing buffer management algorithms demonstrate their blank in leveraging data access features to predict data attributes. In various real-world workloads, most of large sequential write requests are rarely rewritten in near future. Once these write requests occur, many hot data will be evicted from DRAM into flash memory, thus jeopardizing the overall system performance. In order to address this problem, we propose a novel large write data identification scheme, called Prober. This scheme probes large sequential write sequences among the write streams at early stage to prevent them from residing in the buffer. In the meantime, to further release space and reduce waiting time for handling the incoming requests, we temporarily buffer the large data into DRAM when the buffer has free space, and leverage an actively write-back scheme for large sequential write data when the flash array turns into idle state. Experimental results demonstrate that our schemes improve hit ratio of write requests by up to 10%, decrease the average response time by up to 42% and reduce the number of erase operations by up to 11%, compared with the state-of-the-art buffer replacement algorithms.

Keywords SSDs, storage system, buffer management, sequential write requests

Received July 10, 2015; accepted February 29, 2016

E-mail: {zhouwen, dfeng, csyhua, jnliu, huangfangting, chenyu0713}@hust.edu.cn; {zhangshuangwu25}@gmail.com

1 Introduction

Flash-based storage devices have been widely used in industrial PCs and embedded systems for their advantages such as high performance, low energy consumption, light weight and shock resistance. Flash memory stores data in individual memory cells, which are made of floating-gate transistors. With no mechanical components and faster seek location speed, flash-based SSDs have superior read and write I/O performance than conventional hard disk drives (HDDs), especially for random read operation. Compared with HDDs, SSDs have different characteristics in terms of physical structures and data access patterns. Existing file systems and database systems designed for disk-based storage systems fail to be directly used in the flash memory storage systems. To address this problem, Intel takes the lead in adopting the flash translation layer (FTL) [1] between the host interface and the raw flash memory to export SSDs as a block device, thus overcoming the physical restriction of flash memory. With better performance than HDDs, SSDs are more appealing in enterprise servers.

Compared with conventional storage technologies, flash memory has four individual features which motivate researchers to design flash-based management algorithms play-

ing the role of FTL in SSD controller.

- Flash operation commands There are three basic operations for flash memory: read, program and erase. Program operation is served for write requests and alters the data from "1" to "0". The atomic read/program unit is a page which consists of several sectors (512 byte). In general, the capacity of a page is 4KB, 8KB, 16KB or larger. While the length of host requests is variable, varying from one to more pages, a read/write request is translated into one or more pages of read/write operations and the sector address should be aligned with the page address. Erase command is a new operation which reverses "0" to "1". The minimum erase unit is block, which typically contains 16–64 pages. Thus the valid pages must be migrated to other blocks before erasing the candidate block.
- Asymmetric read/write latencies For flash memory, program operation needs to flip bits, consuming longer time and more energy than read operation. In general, the latency of program operation is 10 times longer than that of read operation. Therefore, it is crucial to reduce the number of program operations and enhance the internal parallelism to improve write performance via advanced commands, such as multi-plane, multi-LUN and Copyback [2].
- Out-of-place update Since program operation is unable to reverse the cell bits from "0" to "1", serving an erase operation before program operation is satisfying in a block. In the meantime, out-of-place update leads to frequently changing in the relationship between the logical page number (LPN) and the physical page number (PPN), then an address mapping table is required. Recently, many address mapping algorithms were studied to reduce the size of mapping table and improve the performance of address translation [3,4].
- Endurance limitation The write cycle of each cell is affected by manufacture process. For example, the endurance in the MLC flash memory are typically limited to 10 000 while that is only 3 000 for TLC flash memory. With increasing capacity per chip, the endurance of each cell becomes lower. The endurance problem has greatly inhibited SSDs from being widely used in write-dominant applications. Therefore, many wear-leveling algorithms are researched for SSDs to balance write traffic among the blocks [5,6].

Among the four features described above, the drawbacks of long program latency and low endurance restrict the use of SSDs. Consequently, the flash-aware write-buffer management has been one of key approaches to reduce the number of flash program operations, which enhances both the performance and the lifetime of SSDs.

Buffer management algorithms are widely studied in the databases and operating systems. The target is to evict the page whose next use will occur farthest in the future. Although the existing cache management algorithms can manage the data in the buffer and deploy an appropriate replacement scheme, these algorithms are subject to the defects of being unawareness of the access patterns. We attempt to exploit characteristics of request sequences and predict cold/hot attribute of each page. Therefore, this paper focuses on effectively improving buffer performance through wise buffer management.

SSDs have the potential to replace or complement HDDs in many high-performance demand applications, such as digital city, live map, videos and social network websites. Under these scenarios, millions of multimedia images and videos need to be uploaded and stored in SSDs every day. However, based on our experimental observations, those applications would generate plenty of large sequential write requests, which are less likely to be modified in the near future. We refer to those data as large write data (LWdata) in this paper. In our trace analysis, LWdata spans a wide range of space, whose accumulative size is considerably larger than the buffer capacity. Therefore, once large sequential write requests arrive, such a situation will appear: LWdata will occupy plenty of DRAM space, thus the original data retained in cache will be ejected. It not only brings about drop of buffer hits, but also leads to additional expensive write operations, and reduces lifetime of SSDs. Unfortunately, existing SSDs buffer management schemes simply store the data to meet large-sized requests for the flash memory. However, LWdata generated by many small-sized requests would be written into DRAM buffer, and flush out the previous data in buffer, including hot data to be accessed in future.

Our scheme exploits locality in a different way with existing SSD buffer algorithms. Previous algorithms leverage spatial locality to gather requests from the same physical block in buffer and reduce write amplification by collecting write requests to flash memory. However, our algorithm exploits spatial locality to address buffer pollution induced by LWdata pouring. Existing file systems prevent large write data residing in main memory, while SSD controller fails to identify this LWdata in internal buffer due to the absence of its attribute after passing the device interface.

In this paper, we make the following contributions.

First, in order to address the problem of cache pollution by LWdata, we establish a block-level request sequence model, called R-SEQ, and propose the Prober approach to identify large sequential write request at early stage and label it as cold page in buffer. Incorporated with existing buffer management scheme, SSD controller will migrate all the cold pages to the tail of the queue and discard them from buffer preferentially.

Second, in order to further lessen the response time, we propose an actively write-back scheme. The proposed approach can write back LWdata in background and free up space for incoming requests, which reduces average response time dramatically.

Third, we implement Prober module and actively write-back scheme on SSDsim. Extensive experiments under various real-world workloads with diverse ratio of write requests are conducted to examine write buffer hit ratio, average response time and the number of erase operations. The experimental results reflect that Prober is able to identify cold data accurately and our schemes are very effective.

The rest of this paper is organized as follows. Section 2 gives the background of SSDs and motivation of Prober. Section 3 shows the details of Prober-based buffer management and Section 4 shows the extensive evaluation of Prober. Section 5 summarizes related work. Finally, in Section 6, we conclude the paper.

2 Background and motivation

In this section, we present the background on SSDs to facilitate our discussion and analyze the important observations that motivate our Prober design.

2.1 SSD overviews

Flash memory are widely used as solid-state drives (SSD) on conventional machines. In the internal of SSD, FTL plays an important role in managing DRAM buffer and flash memory. In general, FTL consists of address mapping, wear leveling, garbage collection and buffer replacement components. Some optional functions, such as data compression/encryption and bad block management, are also applied in FTL. Among them, address mapping scheme is the most important one and the foundation of other algorithms.

According to mapping granularity, address mapping algorithms are classified into page-level, block-level and hybrid mapping. For page-level mapping [3,4], a given logical page number (LPN) can be mapped to arbitrary physical page number (PPN) in the flash memory. However, the table size of page-level FTL is too large to be fully stored in the buffer, which leads to severe performance degradation. To reduce the space overhead of the address mapping table, block-level FTL are proposed with a coarse-grained mapping pattern, which records the mapping from logical block number (LBN) to physical block number (PBN) by ensuring the offset in the physical block is the same as that in the logical block. Block-level FTL shows lower space overhead than that of page-level FTL. However, the update operations generated by rewritten requests incur plenty of page migration and consume unaffordable latency. To overcome their respective defects, scientists propose a hybrid mapping scheme which divides the entire blocks into data blocks and log blocks and adopts the page-level and block-level mapping schemes in different area. Log blocks which store frequently updated pages use page-level mapping, while data blocks store consecutive data and adopt block-level mapping. Since log blocks are consumed rapidly in random write applications, hybrid FTL consumes longer time on transforming log blocks to data blocks, which significantly degrades the performance. All kinds of FTL algorithms are well studied to overcome their flaws and improve storage performance.

In addition, another important component of FTL is the buffer management module, which is widely studied in the databases and operating systems. The size of each buffer entry can be a page or a block, which is decided by address mapping granularity. For page-level FTL, DRAM buffer manages each page independently, thus it accommodates more hot pages, achieving higher write performance with random access pattern. While for the block-level FTL, pages with same logical block address are gathered in buffer and written into a physical block together when they are evicted from buffer. It demonstrates higher write performance with sequential access pattern. In this paper, we adopt page-level granularity in DRAM buffer for our research. However, our scheme can also be used in block-level FTL.

Given that asymmetry read/write latencies, the DRAM buffer is mainly used to store dirty data generated by write requests. Our scheme is based on pure write buffer, and it is easy to be transplanted to hybrid buffer.

2.2 Research motivations for Prober

Operations associated with large files are ubiquitous in various applications. On personal desktops and tablet computers, there are frequent file copy operations such as downloading

a video or game from the Internet, copying software from removable drive and creating an image file in local host. On enterprise server, there are also plentiful files uploading operations every day.

One main function of file systems is to translate the write or read operations of the file-level into write or read operations of the block-level. File systems manage metadata and data of files. The block allocation seeks the vacant inodes from metadata area and then writes the vacant inodes directly to the unoccupied data area. Modern file systems employ various techniques like extents, multi-block allocation, delayed allocation and persistent pre-allocation to ensure the efficiency of the allocation and continuity of each sub-block. As large file storage writes plenty of metadata and data to devices, it will generate large sequential write requests at the block-level according to block allocation policy. Large files, once been written to SSDs, will be modified rarely in the near future unless their data are deleted and other files are written to the same logical address. Frequently, modification data are mainly derived from metadata and small files, while these modification operations generate small random write requests in block device.

To observe the real block-level time-space diagram of large file write requests, we conduct the experiment on a RHEL6 Operating System with ext4 file system, an Kingston SSDNow SVP200S3 SSD, build a Samba server, and trap the block-level trace record using the blktrace tool[1)]. Figure 1 depicts the time-space diagram when two remote users upload files in the same local area network (B is an amplification of A). The time-space diagram of the large write requests exhibits great spatial locality. However, existing scheduling schemes fail to identify this characteristic of large write requests, thus resulting in inefficient use of buffer space. This characteristic, once exploited, can efficiently boost the buffer hits, reduce the number of program operations and increase the lifespan of SSDs, which eventually reach an overall performance improvement.

However, in practical applications, the access patterns of real-world workloads will be much more complex than the one that has been depicted in Fig. 1. Figure 2(a) shows the access pattern of RAD-AS [7], which is responsible for worldwide corporate remote access and wireless authentication. The workload covers an 18-hour period and is broken into 1-hour intervals. Figure 2(b) shows a local amplification of the write stream (C) in Fig. 2(a). Obviously, the subfigure shows that RAD-AS consists of many small-sized requests which form a large sequential write sequence. In the meantime, the write requests generated by other threads may form another sequence in different space concurrently.

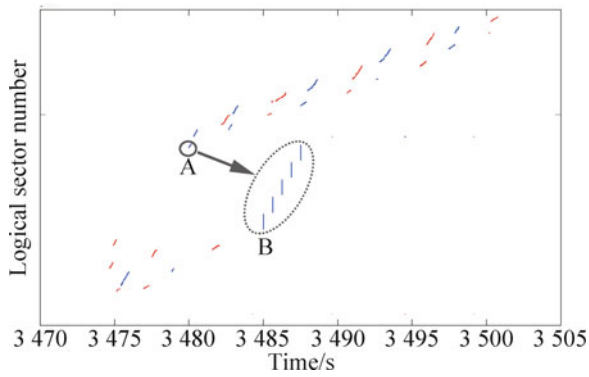


Fig. 1 The time-space diagram of two remote users copying files simultaneously

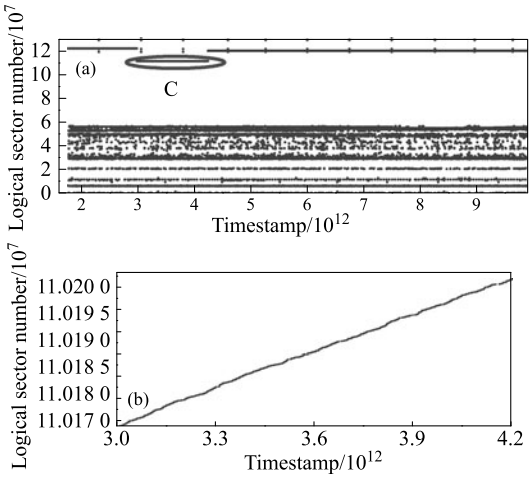


Fig. 2 The time-space diagram of RAD-AS trace. (a) The complete time-space diagram of RAD-AS trace; (b) The local amplification of sequence C

Similar with RAD-AS, other workloads are discovered to contain plenty of sequential write requests as well. Table 1 shows that large sequential write requests with cumulative size exceeding 128KB take a large proportion in all write streams of four workloads. Since the average length of each request is far less than 128KB, the LWdata is usually composed of many small-sized sequential requests. All the workloads used in this paper are available on the Storage Networking Industry Association's trace repository[2)].

Although there is an important characteristic for large sequential write requests, existing buffer management algorithms fail to recognize it. For single queue pattern, buffer replacement policy will select plenty of data from the tail of the queue to evict from buffer, and insert LWdata to the head of the queue, as shown in Fig. 3 ①. For multiple queue pat-

[1)] http://www.cse.unsw.edu.au/ aaronc/iosched/doc/blktrace.html
[2)] http://iotta.snia.org/

tern, buffer replacement policy will select plenty of data from cold queue to evict from buffer and insert LWdata to the head of the cold queue, without affecting the data of hot queue, as shown in Fig. 3 ③. To avoid the LWdata flushing buffer which leads to cache pollution, we insert the requests into the tail of the queue or bypass the requests from buffer directly after they are identified as LWdata, as shown in Figs. 3 ② and 3④.

Table 1 Characteristics of four real-world workloads

Workloads	Avg read size/KB	Avg write size/KB	Write request/%	LWdata/% (ssp = 128KB)
RAD-AS	8.2	9.3	99	64.6
RAD-BE	106.0	11.6	81.7	74.2
MSNCFS	8.6	12.6	26.2	64.3
DAP-DS	31.5	7.0	10	96.2

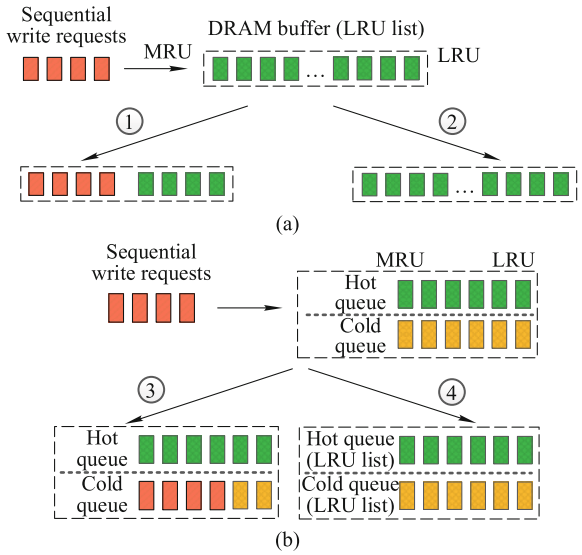


Fig. 3 The influence of large sequential write requests on different buffer replacement policies. (a) Single queue pattern; (b) multiple queue pattern

Experimental results show that evicting LWdata is able to increase the buffer hits. It is indispensable to trap and discard large files data in SSDs buffer at early stage. However, in some sophisticated environments, large sequential write streams are drown in the large amounts of write requests. Specifically, for some complicated applications where exist a lot of large sequential write requests, trapping the large write requests and dealing with them separately can lead to a leap in performance. We propose the novel component called Prober to address this problem.

3 Prober-based buffer management

As shown in Fig. 4, a Prober-based SSD is composed of three main components: Prober-based buffer management module, flash translation layer (FTL) and physical flash memory. Prober sits on the top of FTL inside the SSDs, monitors and processes the I/O requests issued by the upper file system or database system. Prober is used to trap large sequential write sequences. We combine Prober component with flash-aware buffer replacement algorithms, and introduce actively write-back scheme for LWdata.

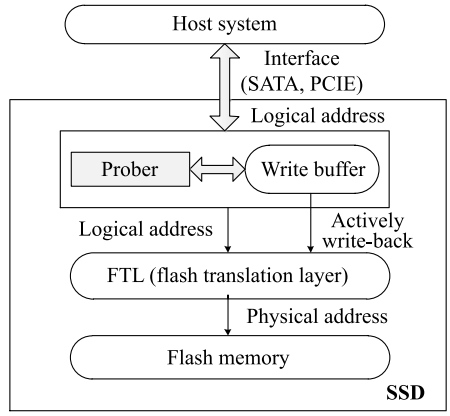


Fig. 4 The architecture diagram of a Prober-based SSD

3.1 Design and implementation of Prober

The write stream consists of many single requests that have three basic attributes: arrival time (*atime*), address of the starting logical sector number (*slsn*), and address of the ending logical sector number (*elsn*), all of which constitute a triple (*atime*, *slsn*, *elsn*) to identify the request. Various applications have different intensities of the block requests. In the case of copying files online from a remote server, we need to take network latency into account. In order to unify measure time, we use arrival serial number (*asn*) to substitute the arrival time. We refer to this model as R-SEQ model. As shown in Fig. 5, there are ten different requests (REQ1 REQ10), each of which is represented by a triple (*asni*, *slsni*, *elsni*), where *i* denotes the request number.

An integrated workload which consists of many requests may exist numerous sets of sequential write sequences. Each set is composed by a group of sequential write requests. The set of large sequential write sequences must meet two conditions. First, the adjacent requests must be spatially sequential. Second, the accumulative size of all requests in a set exceeds threshold which is represented by the set size parameter (*ssp*). In order to restrict the maximal length of queue and reduce temporal and spatial overhead, we only record the address and the arrival time of each request. In addition, we remove the outdated sets beyond threshold which is represented by recent interval parameter (*rip*).

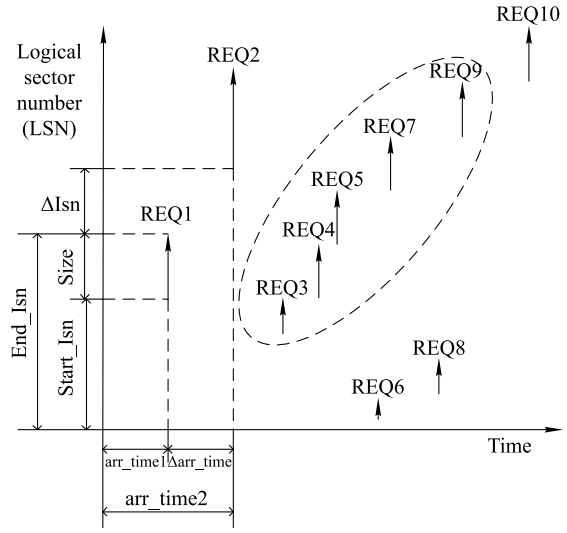


Fig. 5 The request sequence model of R-SEQ

As shown in Fig. 5, if the set of the write stream {REQ3, REQ4, REQ5, REQ7, REQ9} is a large sequential sequence, all the data of {REQ3, REQ4, REQ5, REQ7, REQ9} are considered to be LWdata. Once the incoming request REQ10 is spatially sequential with REQ9, the set of {REQ3, REQ4, REQ5, REQ7, REQ9, REQ10} becomes the substitutes of the original set and is grouped into a new set of large sequential write sequence. Prober algorithm uses a large write request stream queue called LWRS queue to store the sequence numbers of the recently arrived requests. Each node represents a set of sequential write request stream and consists of group (*asn*, *slsn* and *elsn*). *slsn* represents the minimal logical sector number of the set and *elsn* represents the maximal logical sector number of the set. *asn* is the latest updated time of the set. The group (*asn*, *slsn* and *elsn*) changes when a new request inserts the set. Thus we can distinguish multiple sets of large write sequences concomitantly. Any two nodes in LWRS queue are not space intersection. In order to accelerate the querying efficiency, all nodes are sorted according to the ascending order of *slsn*. The maxlength of LWRS queue is limited by *rip*. If any two nodes are spatially adjacent, they will be merged to a new node. Therefore, there is no spatially adjacent node.

The temporal overhead (*Ot*) and the spatial overhead (*Os*) of LWRS are shown as follows.

$$Ot = N \times rip,$$

$$Os = rip \times 16\text{byte},$$

where N represents the number of requests. Each node of LWRS queue takes up 16Byte. *Ot* and *Os* is decided by parameter *rip*.

The structure of LWRS queue is shown in Fig. 6. The detailed workflow of Prober algorithm is described as follows.

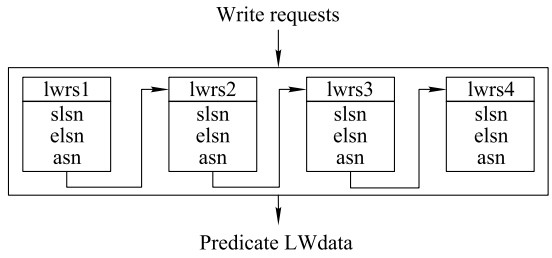


Fig. 6 The management queue of Prober (LWRS queue)

First, Prober scans the LWRS queue and removes all the nodes that are spatially overlapped with the just arrived request or the outdated nodes whose updated time exceeds the predefined threshold (*rip*). Second, Prober inserts the new node that records basic attributes of the just arrived request into the LWRS queue. Third, Prober merges the node with adjacent nodes in the LWRS queue in case they are spatially sequential. Finally, Prober predicates whether the new node belongs to large sequential write stream according to its set size.

3.2 Single queue management with Prober

In our design, Prober is used to identify LWdata and label them as cold pages in SSDs buffer. Thus buffer management algorithm migrates cold pages to the tail of the queue and evicts cold pages from buffer preferentially. Most of existing flash-aware buffer management policies use single LRU queue pattern. According to the characteristics of requests, we partition LRU queue into large data region (LDR) and regular data region (RDR), as shown in Fig. 7. RDR stores regular pages and is followed by LDR which stores LWdata. RDR is organized on the traditional LRU pattern, whose queue head holds the pages with the most frequently accessed. However, LDR is organized on the stack pattern whose queue tail holds the latest accessed pages. To distinguish LDR pages and RDR pages in a single queue, we leverage a flag to label each node. To accommodate flash-aware replacement

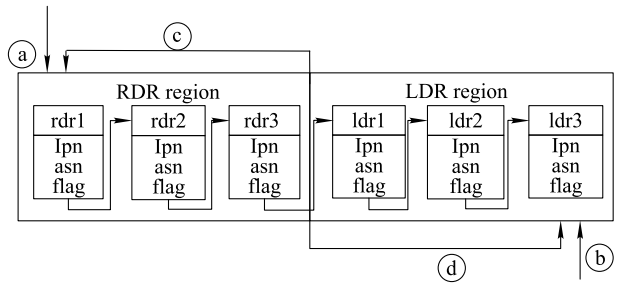


Fig. 7 Prober-based LRU buffer management

algorithm in LDR queue, we introduce *asn*, which represents arrival time of LDR requests. The detailed algorithm is shown as follows.

If a new request Q arrives, Prober first checks whether it belongs to a set of large sequential write sequence and then updates *LWRS* queue according to Prober algorithm. If the request Q belongs to a set of large sequential write sequence, Prober migrates the hit pages or inserts the miss pages into the tail of the LDR queue. The workflow is shown in Figs. 7ⓓ and 7ⓑ. Otherwise, Prober migrates the hit pages or inserts the miss pages into the head of the RDR queue. The workflow is shown in Figs. 7ⓒ and 7ⓐ. It's worth noting that if a set is predicated to large sequential write sequence for the first time, Prober migrates all the pages belonging to this set to the tail of the LDR queue.

Once a SSD works for a long time, the buffer will be overflowed soon. Hence, we need to evict cold pages from buffer preferentially. According to our analysis, the pages in LDR queue are less modified than that in RDR queue in future. For this reason, we first discard pages from the LDR queue and then evict pages from the tail of the RDR queue.

3.3 Multi-queue management with Prober

In some high-end environments, with strong computation capability and high performance requirements, buffer management algorithms are more complex and flexible to accommodate various workloads. In general, the buffer queue is divided into multiple queues according to accessing frequency.

Pages accessed two or more times are resided in hot queue, while the once accessed pages are placed in cold queue. Once the buffer overflows, the replacement policies discard the tail pages of the cold queue preferentially. When the pages of the cold queue are accessed again, buffer management policies migrate them to the hot queue. Usually, the ratio of the capacity between the hot queue and the cold queue remains at a relatively stable proportion. If the size of the hot queue exceeds a predetermined capacity, the tail pages of the hot queue are migrated to the cold queue.

Although multi-queue management schemes prevent cold pages residing in buffer too long, they also consume buffer resource and lead to poor hit ratio. As shown below, we compare Prober-based multi-queue buffer management with one of the best page replacement policies LIRS [8] to show its efficiency.

According to inter-reference recency, LIRS algorithm splits buffer queue into S queue and Q queue. Each queue is managed according to LRU pattern. Q holds the first accessed pages whose state is high inner-reference recency (HIR). While queue S maintains frequently-accessed pages whose state is low inner-reference recency (LIR). Considering the recently-evicted pages may be accessed in near future, LIRS maintains a recent record of the evicted pages with the state of HIR_no_resident, and migrates these pages to the head of S queue if accessed again, preventing the situation that the interval of the frequently-accessed cyclic data set is greater than the buffer capacity.

LIRS discards once-accessed data preferentially. As the amount of LWdata is too large, LIRS is unable to solve cache pollution problem. LWdata leads to the pages in queue Q flushing out from buffer frequently. Thus, the recently-accessed pages which will be accessed again is hard to be moved to queue S. It significantly reduces the hit ratio of queue Q and queue S. To prevent the LWdata from polluting the cache, we use Prober to identify LWdata and evict them from buffer preferentially.

Similar to single LRU queue, queue Q is divided into RDR queue and LDR queue. If a request belongs to a set of large sequential write sequence, we migrate hit pages or insert miss pages into the tail of the LDR queue. The workflow is shown in Figs. 8ⓔ, 8ⓕ, and 8ⓐ. Different with the single LRU queue, if a request do not belong to any set of large sequential write sequence, we migrate hit pages to the head of queue S or insert missed pages into the head of the RDR queue, as shown in Figs. 8ⓒ and 8ⓑ. If the accumulative size of queue S exceeds the predefined capacity, the tail pages of queue S will be migrated into the head of queue Q, as shown in Fig. 8ⓓ. Without Prober component, all the data arrived for the first time will be inserted into the head of the RDR queue. However, our scheme detects LWdata, inserts them into the tail of the LDR and prevents them from polluting the cache.

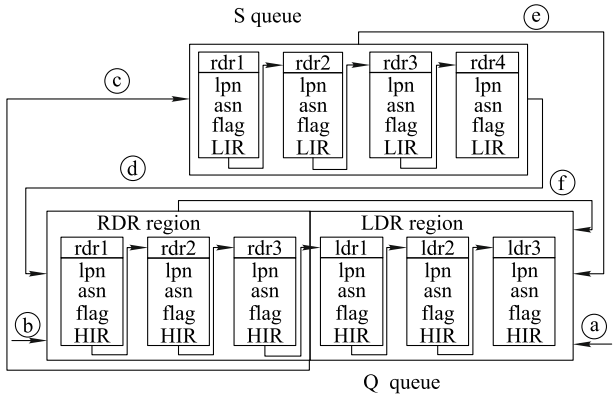


Fig. 8 Prober-based LIRS buffer management

Buffer replacement policy will be activated when buffer

overflows. We select pages from LDR queue as victims preferentially. When the LDR queue is empty, we evict the tail pages of the RDR region.

3.4 Actively write-back scheme

In many applications, the density of the arrival requests is heterogeneous. In the interior of SSDs, the flash array and internal bus are in idle state between two adjacent requests in most cases. Thus we introduce actively write-back scheme to write LWdata into flash array in background. It can free DRAM space and reduce waiting time for incoming requests. Traditional actively write-back scheme which first discards the tail pages of the queue will increase the number of flash program operations and sacrifice longevity since the pages in the queue tail may be accessed in the near future. In our design, we first write-back outdated pages from the LDR queue to overcome these defects. Compared with traditional write-back scheme, our actively write-back scheme reduces response time of write requests dramatically without degrading buffer hits.

In order to implement actively write-back module, we put the LWdata into channel request queue. After the requests complete, we remove relevant pages from buffer. While the LWdata hits before such an operation is completed, we remove the requests from channel queue. Considering the basic unit of request is a sector, a series of continuous and non-full-page large write requests will incur program flash repeatedly. We discard outdated pages in the LDR queue and prevent a non-full page rewriting flash repeatedly.

4 Performance evaluation

4.1 Experimental system

We implement Prober component on a trace-driven SSD simulator called SSDsim [9], which is a real prototype of SSD implementation that has the advantages of statistical energy consumption and simulation flash advanced command. SSDsim is able to emulate most SSDs hardware platforms, mainstream FTL schemes, buffer management algorithms and request scheduling policies. We use page-level FTL policy and simulate a 512GB SSD. To overcome huge space consumption of the address mapping table, we leverage pre-scanning trace approach to hold essential mapping items in memory before the formal process. To evaluate the effect of conventional buffer management incorporated with Prober module, we enhance the most widely used replacement algorithm (LRU) and one of the best page replacement algorithms (LIRS) with Prober in SSDsim. In the meantime, we enhance channel scheduling policy by cooperating with actively write-back module. According to traditional design, we set the ratio of LIR queue capacity to 0.99 and the rest is used for HIR queue. Table 2 shows the parameters used in the simulator. All the workloads used in our experiment are shown in Table 1. Those workloads are mixed with sequential and random data accesses. The proportion of write requests varies from 10% to 99%.

Table 2 Setting parameters used in our simulator

Parameter	Configuration
SSD Capacity	512GB
Redundant space	30%
GC threshold	0.3
Organization	@4 channels, 4 chips/channel, 4 dies/chip 2 planes/die, 4096 blocks/plane 256 pages/block, 8KB per page
Timing characteristics	@tBERS=1.5ms, tPROG=200us, tR=20us tWC=25ns, tRC=25ns

In the following evaluation, we use hit ratio of read and write requests, average response time and the number of erase operations as performance metrics. The hit ratio refers to the ratio of read and write requests whose data can be hit in buffer. If a write request misses in buffer, it will evict extra pages from buffer and lead to a flash program operation. Once a read request misses in buffer, it need to read flash directly. Average response time of all requests refers to the time period between its arrival and completion, containing waiting time. This time may be affected by garbage collection operation. The number of erase operation can be used to measure the service lifetime of SSDs. The smaller the number of erase operation is, the longer lifetime of SSDs achieves.

4.2 Evaluation results

4.2.1 Parameter training (*rip* and *ssp* vary with buffer capacity)

As the parameters of *rip* and *ssp* are independent, we study them respectively. In the experiments, we compare Prober-based LRU and LIRS algorithms with pure LRU and LIRS algorithms. In our parameter training test, we do not consider the effect of buffer hits on actively write-back module.

1) Parameter *ssp* training

Figures 9 and 10 plot the hit ratio of write requests under Prober-based LRU algorithm normalized to pure LRU algorithm as a function of the DRAM capacity. The curves exhibit similar trend with various buffer capacities. In the initial

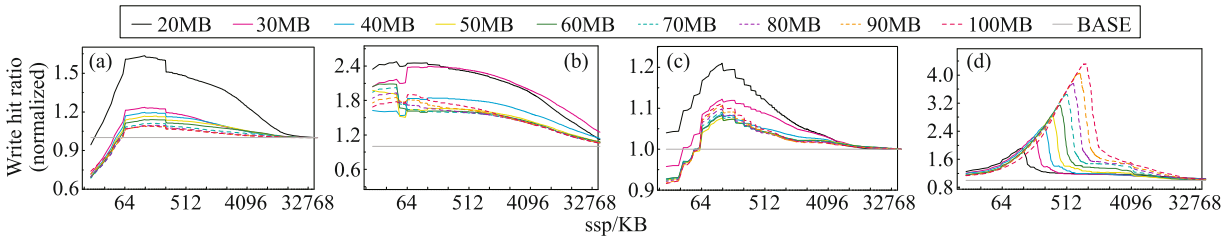


Fig. 9 Hit ratio of write requests on Prober-based LRU algorithm normalized to pure LRU algorithm. (a) RAD-AS; (b) RAD-BE; (c) MSNCFS; (d) DAP-DS

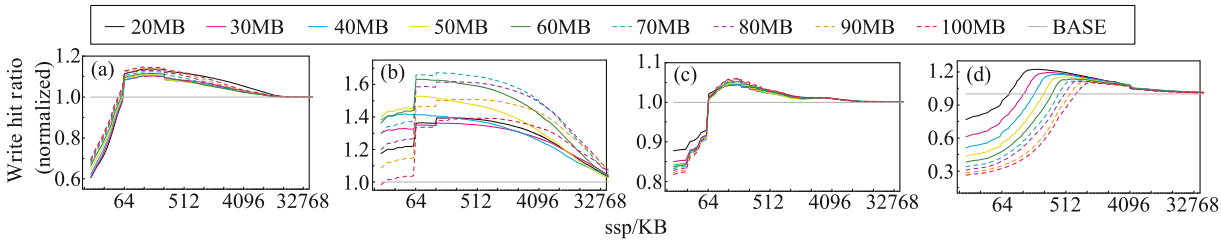


Fig. 10 Hit ratio of write requests on Prober-based LIRS algorithm normalized to pure LIRS. (a) RAD-AS; (b) RAD-BE; (c) MSNCFS; (d) DAP-DS

stage, write hits ascend with the increase of *ssp* since regular write request streams have been filtered. When write hit ratio reaches peak value, *ssp* reaches the optimal value. As *ssp* continues to increase, hit ratio starts to descend because Prober is unable to distinguish small-sized LWdata. Finally, write hit ratio tends to be stable and close to that of pure LRU/LIRS algorithms after *ssp* reaches to 32MB.

For RAD-AS, RAD-BE and MSNCFS workloads, the optimal values of *ssp* are 128KB, and their write hit ratios reach peak values. Experimental results show that the smaller DRAM capacity is, the more hit ratio of write requests improves. As DRAM capacity equals to 20MB, Prober improves the write hit ratio up to 63.6%, 145%, and 20.9% higher than that of pure LRU algorithm, and up to 13.6%, 39.5%, and 4.4% higher than that of LIRS algorithm. However, DAP-DS trace exhibits very different features with other traces, the optimal value of *ssp* increases progressively along with the buffer capacity. Although hit ratio with DAP-DS workload improves the most, it is difficult to select a stable *ssp* value for the optimal performance.

From the results we can conclude that the smaller buffer capacity is, the more effective Prober will be. Moreover, Prober can improve more when write hit ratios of pure LRU/LIRS are lower. To evaluate other parameters under the optimal value, we select RAD-AS, RAD-BE, and MSNCFS workloads to verify the whole performance with a constant *ssp* value of 128KB.

2) Parameter *rip* training

To get optimal values of *rip* more stably, we use a 100MB DRAM for the experiment of *rip* training. Figure 11 shows the hit ratio of write requests with various of *rip* using three

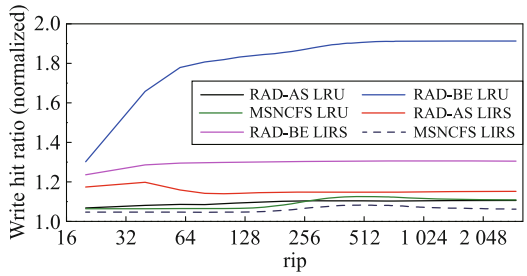


Fig. 11 The influence of parameter *rip* on hit ratio of write requests

real-world workloads. We can find out that hit ratios of enhanced LRU and LIRS schemes gradually grow following the increasing of *rip*. As the length of *LWRS* queue is limited by the value of *rip*, Prober traps very few LWdata when the value of *rip* is small. With *rip* growing from 20 to 512, the normalized hit ratio increases considerably. After *rip* exceeds 512, the hit ratios of LRU and LIRS algorithms tend to be stable and close to the peak value. However, the continual increasing of *rip* brings more hardware overhead. Therefore, we select an affordable value 3 000 for the following experiments. Under this, Prober needs spatial overhead of 46.9Kbyte, which remains constant.

4.2.2 Overall performance

In the whole system tests, we evaluate the affection of actively write-back scheme on buffer hit ratio of read and write requests, average response time and number of erase operations with three workloads. In the meantime, we test all as-

pects of the performance with Prober and actively write-back modules with various buffer capacities.

As shown in Figs. 12(a)–12(c), we evaluate the efficiency of actively write-back scheme by comparing it with inactively write-back scheme which adopts Prober schemes but does not write LWdata into flash in background. In the write hit ratio and the number of erase operation tests, the performance of actively write-back scheme is better than pure buffer replacement schemes and the same as inactively write-back scheme. It illustrates the LWdata discarded from buffer in background is less re-written in the near future. The actively write-back scheme does not affect the lifetime which is very important in the practical situation. Figure 12(c) shows actively write-back scheme reduces the average response time dramatically. Compared with inactively write-back scheme, actively write-back scheme reduces average response time by 27.2%, 16.6% and 5.7%, respectively. In particular, the response time of inactively write-back in the RAD-BE trace is higher than pure LRU despite that the write hit ratio is lower, because the read hit ratio is reduced seriously.

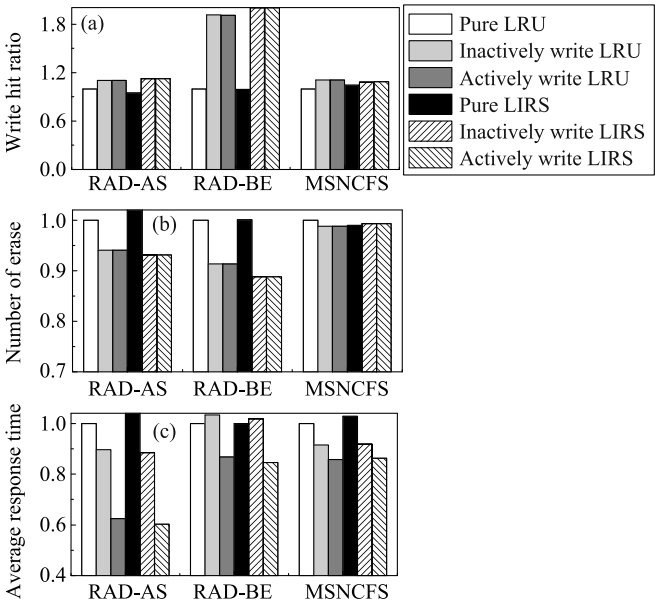


Fig. 12 The influence of actively write-back scheme (normalized). (a) Write hit ratio; (b) the number of erase operations; (c) average response time

Figures 13–16 plot the comprehensive results.

• Hit ratio of read requests Figure 13 shows the hit ratio of read requests with various buffer capacities using three workloads. For RAD-AS and RAD-BE, Prober algorithm reduces the read hits seriously in most cases. As to MSNCFS, the Prober-based schemes have a little improvement on read hits. Consequently, Prober has uncertainty influence on hit ratio of read requests.

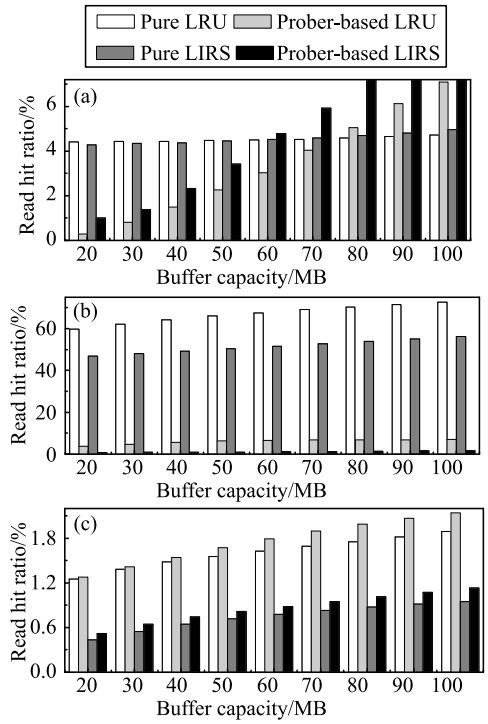


Fig. 13 Hit ratio of read requests under Prober using three workloads. (a) RAD-AS; (b) RAD-BE; (c) MSNCFS

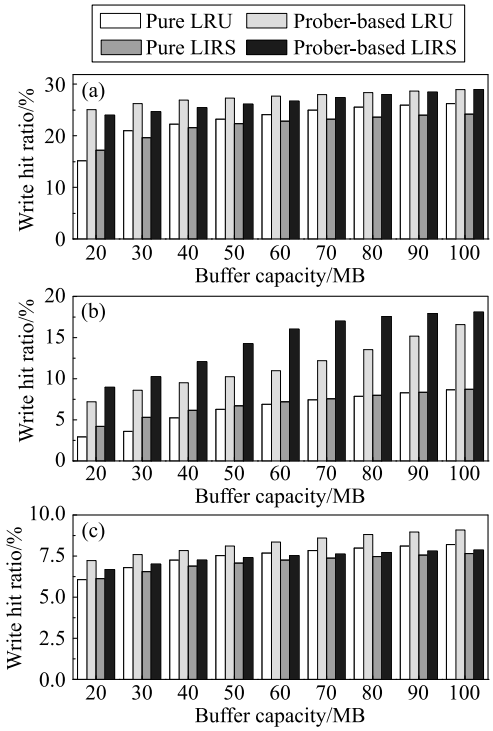


Fig. 14 Hit ratio of write requests under Prober using three workloads. (a) RAD-AS; (b) RAD-BE; (c) MSNCFS

• Hit ratio of write requests Figure 14 shows the hit ratio of write requests with various buffer capacities using three workloads. The hit ratio of pure LIRS algorithm is better than

pure LRU obviously, especially in a small-sized buffer. Incorporating with Prober, buffer replacement algorithms are able to enhance buffer hits more. Prober-based LRU has more improvement than LIRS. For RAD-BE, Prober enhances the buffer up to twice on both LRU and LIRS.

• The number of erase operations Figure 15 shows the number of erase operations as a function with various buffer capacities using three workloads. Erase operation reflects the available lifetime of SSDs. With write hits increased by Prober, the number of program operations declines, hence the longer available lifetime achieves. For RAD-AS and RAD-BE, the erase number is reduced by 10% and 11.2%, respectively. It is worth noting that the performance of Prober-based LIRS is better than that of LRU since it increases write hit ratio greatly and reduces the number of flash program operations. For MSNCFS, the number of erase operations reduces little since the ratio of write requests is very small.

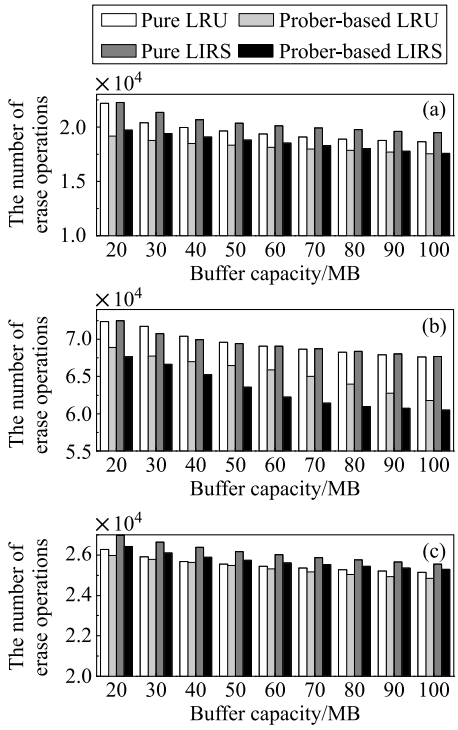


Fig. 15 The number of erase operations under Prober using three workloads. (a) RAD-AS; (b) RAD-BE; (c) MSNCFS

• Average response time Figure 16 shows the average response time as a function with various buffer capacities using three workloads. Average response time is decided by buffer hits and actively write-back scheme. For RAD-AS and RAD-BE, the average response time is affected by the above two reasons, thus the value reduces by 42.6% and 16.6%, respectively. While in the MSNCFS trace, the buffer hits of write requests have no increment on Prober, the average response time reduces by 16% due to the fact that actively write-back scheme frees space for the incoming requests.

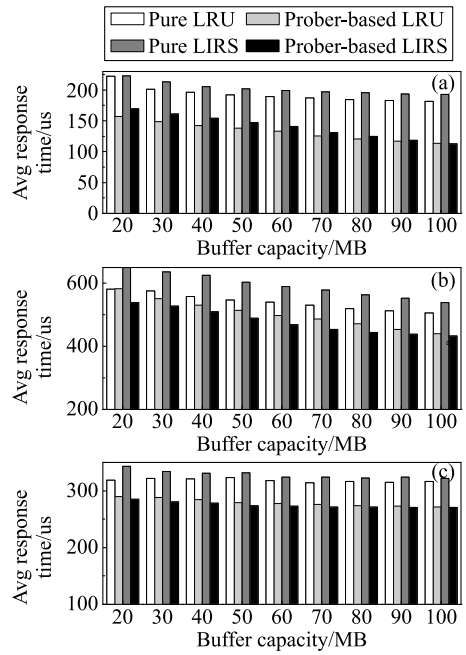


Fig. 16 Average response time under Prober using three workloads. (a) RAD-AS; (b) RAD-BE; (c) MSNCFS

Finally, by analyzing the experimental results, we come to the following conclusion. Prober is able to improve buffer hits of write requests and thus prolongs the lifetime of SSDs, while the actively write-back scheme reduces the average response time of requests dramatically.

5 Related work

5.1 Buffer management policy

The buffer management policies widely used in commercial systems are LRU and its variants. LRU always evicts the least recently used page. Due to LRU's inability to identify access patterns that have weak locality, anterior works have tried to address this important cache pollution problem by using advanced replacement algorithms. LRFU [10] evicts pages which have the smallest CRF value based on accessing time and frequency. This algorithm makes a good trade-off between time and frequency by choosing an appropriate parameter λ. However, as λ is a fixed value, it will incur non-negligible performance degradation when access pattern changes dynamically. MQ [11], LRU-K [12], and LIRS [8] attempt to enhance LRU's capacity by making use of additional historical record of previous block references besides the recent information used in LRU.

Recently, several flash-aware buffer management policies have been proposed to address the asymmetric read/write problem of flash memory. FAB [13], BPLRU [14], and BPAC [15] are three block-level buffer management policies. The target of these algorithms is to achieve better random write performance by changing request stream to provide more sequential writes for flash. They are all designed for the small buffers in embedded flash devices.

More recently, TBF [16] uses two Bloom Filters to maintain the cache objects rather than LRU queue. It achieves a lower spatial overhead and faster inquiring speed, but improves hits little.

Several hybrid management policies, such as Clean first LRU (CFLRU) [17] and LRU-WSR [18], are explored to schedule read and write queues. Clean pages are generated by read requests, and dirty pages are generated by write requests. Due to asymmetric read/write latencies, clean pages are more proper to be replaced. LRU-WSR re-orders cold pages in dirty queue and writes them to flash memory. The enhanced LRU-WSR algorithm focuses on reducing the number of write/erase operations as well as preventing serious degradation of buffer hits.

In order to further improve the performance, CCF-LRU [19], AD-LRU [20], and FD-Buffer [21] preferentially evict cold pages according to access frequency. While FOR [22] introduces frequency and recency to select a more suitable page as a victim.

All the approaches above can identify cold data that have few opportunities to be accessed in future. Different with those approaches, our work focuses on predicting cold data according to the cumulative size of sequential write request stream. Prober is able to incorporate with existing buffer replacement algorithms to identify cold data more accurately.

5.2 Actively write-back scheme

Several previous researches [23,24] leverage idle stage of flash memory to improve write performance via intelligent scheduling. Eager write-back [25] is one of the most classical algorithms. Data in the tail of queue would be written back to flash memory before normal discarding. Unfortunately, the early discarded data may be accessed in the near future for inaccurate prediction policy. Once the early write-back scheme is applied in SSD buffer, it will increase the number of program operations, thus reduces the lifetime of SSDs. Nonetheless, Prober covers the shortage of inaccurating prediction and can be cooperated with eager write-back to achieve better performance.

6 Conclusion

In recent years, SSDs achieve their greatest popularity in consumer electronics and data center, but still restricted by many intrinsic problems, such as long write latency and low endurance. Using DRAM as the internal buffer of SSDs can be an effective solution. However, existing flash-aware buffer replacement algorithms encounter cache pollution problem which is incurred by LWdata.

In this paper, we propose a novel large write data identification scheme, called Prober. By establishing large write sequence model, we trap LWdata in early stage, which effectively prevents those data replacing hot pages from buffer. In the meantime, we introduce actively write-back scheme for LWdata, which frees space for the incoming requests. By the experiment of parameter training, we obtain the range of optimal value, select a reasonable value for the workloads of RAD-AS, RAD-BE, and MSNCFS, and evaluate the performance of Prober and actively write-back scheme.

Our experimental results show that Prober-based buffer replacement scheme is benefit for the lifetime of SSDs, and the actively write-back scheme is conducive for the response time of write requests. Finally, our scheme is efficient to exploit the workloads which contain massive large sequential write sequences, especially in the write-dominant traces.

Acknowledgements This work was supported by the National High-tech R & D Program of China (863 Program) (2015AA016701, 2015AA015301, and 2013AA013203); the National Natural Science Foundation of China (Grant Nos. 61303046, 61402189, and 61173043); State Key Laboratory of Computer Architecture (CARCH201505). This work was also supported by Key Laboratory of Information Storage System, Ministry of Education, China. The SSDsim software used in this paper is licensed under the GNU General Public License and is available for download at http://storage.hust.edu.cn/SSDsim/.

References

1. Intel Corporation. Understanding the flash translation layer (FTL) specification. Application Note AP-684. 1998
2. Micron Technology Inc. 64Gb, 128Gb, 256Gb, 512Gb Asynchronous/Synchronous NAND Features. Micron, 2009
3. Jiang S, Zhang L, Yuan X H, Hu H, Chen Y. S-FTL: an efficient address translation for flash memory by exploiting spatial locality. In: Proceedings of the 27th IEEE Symposium on Mass Storage Systems and Technologies. 2011, 1–12
4. Gupta A, Kim Y, Urgaonkar B. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating System. 2009
5. Chang L P. On efficient wear leveling for large-scale flash-memory storage systems. In: Proceedings of the 2007 ACM symposium on Ap-

plied computing. 2007, 1126–1130

6. Murugan M, Du D H C. Rejuvenator: a static wear leveling algorithm for NAND flash memory with minimized overhead. In: Proceedings of the 27th IEEE Symposium on Mass Storage Systems and Technologies. 2011, 1–12
7. Kavalanekar S, Worthington B, Zhang Q, Sharda V. Characterization of storage workload traces from production windows servers. In: Proceedings of IEEE International Symposium on Workload Characterization. 2008, 119–128
8. Jiang S, Zhang X. LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. In: Proceedings of ACM SIGMETRICS Conference. 2002, 31–42
9. Hu Y, Jiang H, Feng D, Tian L, Zhang S, Liu J, Tong W, Qin Y, Wang L. Achieving page-mapping FTL performance at block-mapping FTL cost by hiding address translation. In: proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies. 2010
10. Kim C S. LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Transactions on Computers, 2001, 50(12): 1352–1361
11. Zhou Y, Philbin J, Li K. The multi-queue replacement algorithm for second level buffer caches. In: Proceedings of the 2001 USENIX Annual Technical Conference. 2001, 91–104
12. O'neil E J, O'neil P E,Weikum G, The LRU-K page replacement algorithm for database disk buffering. ACM SIGMOD Record, 1993, 22(2): 297–306
13. Jo H, Kang J U, Park S Y, Kim J S, Lee J. FAB: flash-aware buffer management policy for portable media players. IEEE Transactions on Consumer Electronics, 2006, 52(2): 485–493
14. Kim H, Ahn S. BPLRU: a buffer management scheme for improving random writes in flash storage. In: Proceedings of the 6th USENIX Conference on File and Storage Technologies. 2008
15. Wu G, Eckart B, He X. BPAC: an adaptive write buffer management scheme for flash-based solid state drives. In: Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies. 2010, 1–6
16. Canim M, Mihaila G A, Bhattacharjee B, Lang C A, Ross K A. Buffered bloom filters on solid state storage. In: Proceedings of VLDB ADMS Workshop. 2010
17. Park S Y, Jung D, Kang J U,Kim J S, Lee J. CFLRU: a replacement algorithm for flash memory. In: Proceedings of the 2006 ACM International Conference on Compilers, Architecture and Synthesis for Embedded Systems. 2006, 234–241
18. Jung H, Shim H, Park S, Kang S, Cha J. LRU-WSR: integration of LRU and writes sequence reordering for flash memory. IEEE Transactions on Consumer Electronics, 2008, 54(3): 1215–1223
19. Li Z, Jin P, Su X, Cui K, Yue L. CCF-LRU: a new buffer replacement algorithm for flash memory. IEEE Transactions on Consumer Electronics, 2009, 55(3): 1351–1359
20. Jin P, Ou Y, Härder T, Li Z. AD-LRU: an efficient buffer replacement algorithm for flash-based databases. Data & Knowledge Engineering, 2012, 72: 83–102
21. On S T, Li Y, He B, Wu M, Luo Q, Xu J. FD-buffer: a buffer manager for databases on flash disks. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management. 2010, 1297–1300
22. Lv Y, Cui B, He B, Chen X, Operation-aware buffer management in flash-based systems. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data. 2011, 13–24
23. Canim M, Mihaila G A, Bhattacharjee B, Ross K A, Lang C A. SSD bufferpool extensions for database systems. Proceedings of the VLDB Endowment, 2010, 3(1–2): 1435–1446
24. Stuecheli J, Kaseridis D, Daly D, Hunter H C, John L K. The virtual write queue: coordinating dram and last-level cache policies. ACM SIGARCH Computer Architecture News, 2010, 38(3): 72–82
25. Lee H H S, Tyson G S, Farrens M K. Eager writeback — a technique for improving bandwidth utilization. In: Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture. 2000, 11–21

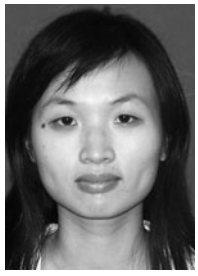
Wen Zhou received the BE degree in computer science and technology from the Huazhong University of Science and Technology (HUST), China in 2009. He is currently working toward the PhD degree in computer architecture from HUST. His research interest includes reconfigurable computing on FPGA and non-volatile memory-based storage system.

Dan Feng received the BE, ME, and PhD degrees in computer science and technology in 1991, 1994, and 1997, respectively, from Huazhong University of Science and Technology (HUST), China. She is a professor and vice dean of the School of Computer Science and Technology, HUST. She has more than 100 publications in major journals and international conferences, including IEEE-TC, IEEE-TPDS, ACM-TOS, JCST, FAST, USENIX ATC, ICDCS, HPDC, SC, ICS, IPDPS, and ICPP. She serves on the program committees of multiple international conferences, including SC 2011, 2013 and MSST 2012. She is a member of IEEE and a member of ACM. Her research interests include computer architecture, massive storage systems, and parallel file systems.

Yu Hua received the BE and PhD degrees in computer science from the Wuhan University, China in 2001 and 2005, respectively. He is an associate professor at the Huazhong University of Science and Technology, China. He has more than 60 papers to his credit in major journals and interna-

tional conferences including IEEE Transactions on Computers (TC), IEEE Transactions on Parallel and Distributed Systems (TPDS), USENIX ATC, USENIX FAST, INFOCOM, SC, ICDCS, ICPP. He has been on the program committees of multiple international conferences, including INFOCOM, ICDCS, ICPP, and IWQoS. He is a senior member of the IEEE, and a member of ACM and USENIX. His research interests include computer architecture, cloud computing, and network storage.

Jingning Liu received the BE degree in computer science and technology from the Huazhong University of Science and Technology (HUST), China in 1982. She is a professor in the HUST and engaged in researching and teaching of computer system architecture. She has over 20 publications in journals and international conferences. Her research interests include computer storage network system, high-speed interface and channel technology, embedded system and FPGA design.



Fangting Huang received the BE degree in software engineering from the Sun Yet-sen university, China in 2010. She is currently working toward the PhD degree in computer architecture from the Huazhong University of Science and Technology, China. She publishes several papers in major conferences including IPDPS, etc. Her research interest includes computer architecture and storage systems.

Yu Chen received the BE degree in computer science and technology from the Huazhong University of Science and Technology (HUST), China in 2013. She is currently working toward the PhD degree in computer architecture from HUST. She publishes several papers in major conferences including DATE, etc. Her research interest includes software-defined storage.

Shuangwu Zhang received the BE degree in computer science and technology from the Huazhong University of Science and Technology (HUST), China in 2012. He is currently a master student in HUST. His research interest includes computer architecture and storage systems.