


SPECIAL ISSUE PAPER

ECR: Eviction-cost-aware cache management policy for page-level flash-based SSDs

Hao Chen¹  | Yubiao Pan² | Cheng Li¹ | Yinlong Xu¹

¹School of Computer Science and Technology, University of Science and Technology of China, Hefei, China

²School of Computer Science and Technology, Huaqiao University, Xiamen, China

Correspondence

Yinlong Xu, School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China.
Email: ylxu@ustc.edu.cn

Funding information

National Nature Science Foundation of China, Grant/Award Number: 61772486 and 61802133; Nature Science Foundation of Fujian Province, Grant/Award Number: 2018J05107

Summary

Cache management policy plays a key role in offering low latency access to flash-based SSDs. Most existing solutions including LRU and its successors only focus on improving the cache hit ratio, but rarely consider to reduce the waiting time of the eviction operation in the page-level mapping FTLs. As the workloads spreading across internal chips of modern flash-based SSDs are often highly imbalanced when workloads are write-intensive, the time cost of evicting a dirty page from cache varies in a wide range. In this paper, we propose a novel eviction-cost-aware cache management policy, called ECR, to minimize the eviction cost in write-dominant applications. ECR gives a higher probability to evict a page, which causes the shortest waiting time in the corresponding chip queue. To achieve this, we introduce a monitor module to keep track of states of all chip queues, and a multi-LRU list structure to accelerate the selection of a victim chip and a target page in cache to perform an eviction. Our experimental results show that ECR can significantly reduce the average response time by as much as 59.55% and 44.84% compared to LRU and GCaR-CFLRU, respectively, where GCaR-CFLRU is the combination of state-of-the-art algorithm GCaR and CFLRU.

KEYWORDS

cache management, flash memory, SSD

1 | INTRODUCTION

NAND-flash-based solid-state drives (SSDs) show superior performance over hard disk drives (HDDs), in terms of lower access latency, smaller size, lower energy consumption, noise avoidance, and shock resistance.¹ In recent years, SSDs have received a great deal of attention from industry and academia. Besides the deployment on personal computers and mobile devices, they also have been widely applied in the high performance computing and enterprise environments.² Moreover, with the drop of per-bit cost, SSDs are expected to be used much more widely.³

In order to reduce the access latency of user's I/O requests, SSDs usually adopt an on-board device cache, such as DRAM or SRAM, to smooth user's I/O requests traffic.⁴ The user's I/O requests from upper-level applications can be kept in cache for as long time as possible before being evicted to the flash chips.³ In other words, the on-board cache plays an important role in offering a better I/O performance since a number of requests can be served in cache.⁵ However, the effectiveness of a cache mainly relies on its replacement policy due to its limited size.^{3,6}

Least Recently Used (LRU) is the most widely used cache replacement algorithm because of its simple and effective exploitation of temporal locality.⁷ Based on LRU, there are some algorithms have been proposed for some particular workloads, such as 2Q,⁸ LRU-K,⁹ ARC,¹⁰ and LIRS.¹¹ However, all of them are not so efficient for SSDs. Flash-based SSDs have some unique characteristics compared to HDDs, such as asymmetric costs of read and write, out-of-place update, garbage collection, and multiple channels connecting to multiple chips. These unique characteristics make the HDD-oriented replacement algorithms not efficient for SSDs.¹² Recently, there are some flash-aware cache replacement algorithms been proposed. For example, CFLRU¹³ divides the LRU list into working region and clean-first region. It prefers to evict clean pages in the clean-first region. LRU-WSR,¹⁴ CCF-LRU,¹⁵ and AD-LRU¹⁶ put efforts to improve CFLRU. GCaR¹⁷ gives higher priorities to cache pages belonging to the flash chips in the GC state to reduce the response delay.

However, existing works are not aware of the states of underlying request queues within flash chips, where queues may have different numbers of pending requests and the completion time of different requests vary a lot, eg, the GC request usually takes much longer than normal read or

write requests. Due to this ignorance, thus, they are not aware of the cost of evicting a dirty page from cache. In write-dominant applications, there are many dirty pages in cache due to frequent write/update in cache. Therefore, it has a very high probability to evict a dirty page from cache when cache is full. Evicting a dirty page will cause a write back operation, which will be regarded as a normal write request enqueued into the corresponding chip queue and waits to be scheduled until all the requests prior to it in the queue are completed. Minimizing the waiting time for write back operations will reduce the eviction cost. To do this, cache replacement algorithms should be aware of the states of queues within flash chips.

In this paper, we propose an Eviction-Cost aware Cache Replacement algorithm, named ECR, to improve the performance of flash-based SSDs. The basic idea is to minimize the waiting time of write back operations when the cache is full. To achieve it, we prefer to evict a dirty page belonging to the chip which needs shortest time to complete all requests in its queue. The main contributions of this paper are as follows.

- We propose a model to estimate the time of a chip completing all the requests in its queue, including the potential garbage collection and metadata writing.
- We propose an eviction-cost-aware cache replacement algorithm, which always evicts the least recently used page in the chip with the shortest waiting time.
- We design a multi-LRU lists structure, in which each LRU list is dedicated to recording all dirty pages on each chip, to speed up the search operation and victim selection in cache. Besides, we keep the clean pages in cache in a *Clean-LRU* list.
- We implement ECR and compare it with state-of-the-art proposed schemes. Our experimental results show that ECR can reduce the average response time by as much as 59.55%, 55.32%, and 44.84% compared to LRU, CFLRU, and GCaR-CFLRU, respectively.

The rest of this paper is organized as follows. In Section 2, we briefly introduce the background and motivate our work. In Sections 3 and 4, we present the detailed design of ECR and the experimental results, respectively. In Section 5, we compare ECR with some related works. We conclude this work in Section 6.

2 | BACKGROUND AND MOTIVATION

2.1 | Background

In this paper, we consider NAND flash memory, or short flash Table memory. A flash chip is divided into some blocks, and a block consists of some pages. A flash page must be erased before writing new data to it because of physical constraints. To improve write performance, flash devices usually implement out-of-place updates, each of which writes the new data to a free page and invalidates the previously mapped page. If the number of free blocks becomes smaller than the predetermined threshold, the garbage collection (GC) will be evoked, which selects a victim block to recycle, ie, reads the valid pages from the block, re-writes them into a free block, and then erases the victim block. The granularity of write/read operations in flash memory is a *page*, which is typically 4 to 16 KB in today's flash devices,¹⁸ while the erase operation must be operated at the granularity of a block. The time cost of an erase operation is usually one order of magnitude higher than a write operation, which can significantly increase the waiting time of use's requests in the queue.

Figure 1 shows the typical architecture of an SSD.^{19,20} When an I/O request arrives, the host interface logic (HIL) in SSD will enqueue it into a device queue and translate it into a flash-aware request. The flash translation layer (FTL) is mainly responsible for cache management, address translation, and I/O scheduling. At the back end of SSDs, there are multiple independent channels, each of which connects to one or more flash chips. Each flash chip consists of several dies, and there are a number of planes in each die.¹⁹ If a request does not hit in cache, the logical page number (LPN) of the request should be translated to its physical page number (PPN). After that, the request will be enqueued into the corresponding chip queue according to its address and waits to be dispatched. In the chip queue, some of the I/O requests come from the host and some of them are imposed by garbage collection. Due to the spatial and temporal locality, it is common that applications frequently access

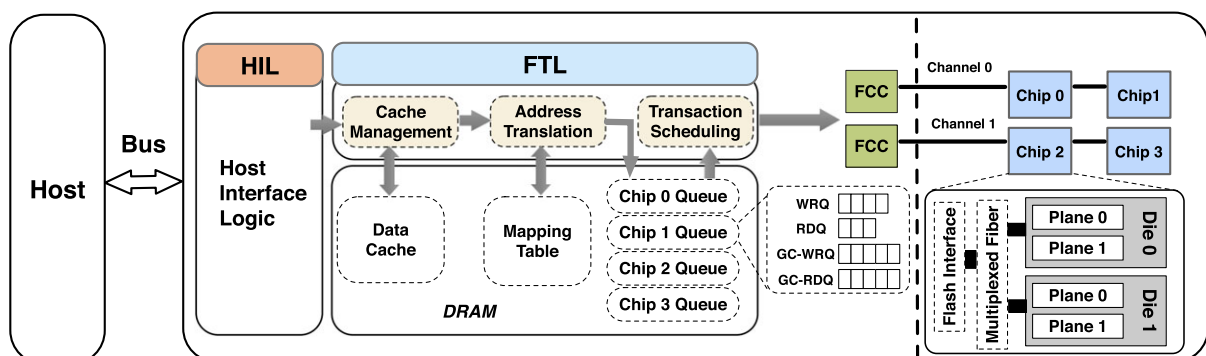


FIGURE 1 Internal organization of SSDs

the same flash chips within a short time interval.²¹ In practical systems, the workloads on different chips may be seriously skewed, which means that some chips are very busy and some are not busy or idle. If a chip is overloaded, it will incur long waiting time for the I/O requests submitted to it and postpone the completion time of the I/O requests.

2.2 | Motivation

Most of the traditional cache management algorithms are designed to increase the hit ratio. However, the hit ratio is very relevant to the cache size and the access pattern of workloads. Therefore, we cannot consistently get a high hit ratio because of the limited cache size and poor locality in some workloads. From another point of view, we can also improve the performance of cache by optimizing the replacement scheme. When the cache is full, we should evict a page in cache to make room for the accessed page. If the evicted page is a dirty page, ie, it has been updated since it was loaded into cache, we should write back the page into flash and induce to a write back operation. In write-dominant workloads, there are many dirty pages in cache such that dirty pages are likely to be evicted. Therefore, reducing the number of evicted dirty pages and minimizing the eviction cost will also improve the performance of cache.

There are some approaches to optimize the replacement scheme in cache. CFLRU¹³ prefers to evict clean pages in LRU list to avoid writing back dirty pages into flash chips. However, in write-dominant workloads, there are not so many clean pages in cache. Due to the combined consideration of both eviction efficiency and hit ratio, CFLRU prefers to evict a clean page in clean-first region. However, it is hard to find a clean page for eviction in write-dominant workloads. To study the eviction efficiency of CFLRU, we conducted experiments with six write-dominant workloads. The experimental setup and the selected traces are presented in Table 2 and Table 3 in Section 4. Figure 2 shows that the ratio of replacing a dirty page with CFLRU is 61.02%-94.42% for six workloads even if it gives a higher priority to evict clean pages. All the CFLRU-based schemes¹⁴⁻¹⁶ encounter the same problem.

GCaR¹⁷ tries to avoid evicting pages to the chips in the GC state and alleviates the contention between write back operations and GC operations. Thus, GCaR reduces the delay of user requests submitted to some chips which are executing GC. In the same experimental setup as in Figure 2, we find that only 19.28% to 67.81% of the evicted dirty pages belong to the chips in the GC state in LRU scheme, as shown in Figure 3. This implies that there are only a small portion of situations that can be optimized by GCaR. Therefore, we can improve GCaR for the case of an SSD being not in the GC state by taking the skewness of loads on different chips into consideration.

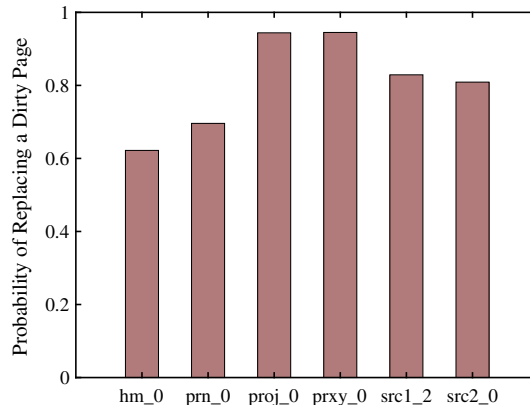


FIGURE 2 The probability of replacing a dirty page in CFLRU

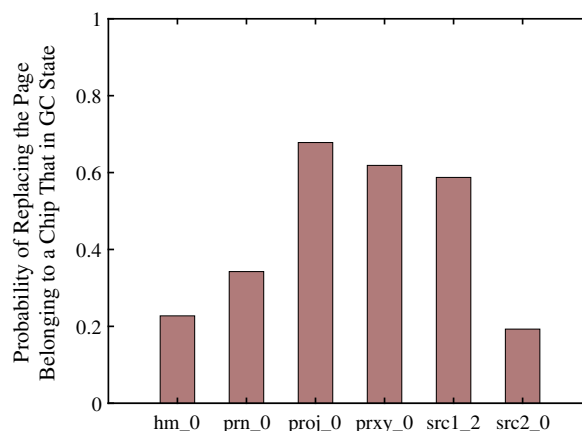


FIGURE 3 The probability of replacing a dirty page belonging to the chip that is in the GC state

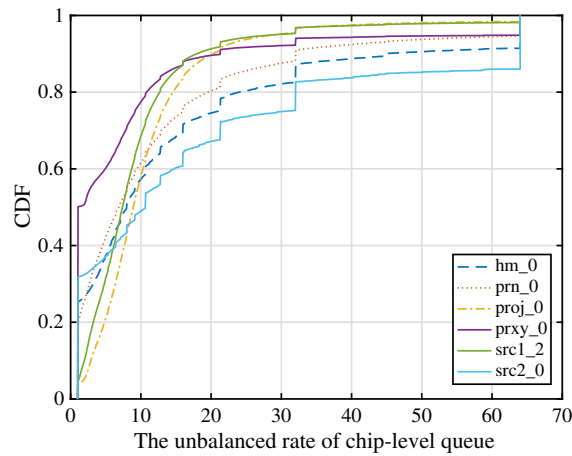


FIGURE 4 The CDF of unbalance rate of chip-level queues

In SSDs, the access delay to chips consists of two parts, the flash access time and the waiting time. The access time for a write/read request is decided by hardware and fixed, while the waiting time is the time duration between a request enqueued into a chip queue and dequeued from it. The waiting time depends on the access time of all the requests in the corresponding chip queue. Reducing the waiting time of a request can not only reduce its response time, but also reduce the waiting time of subsequent requests to this chip. We define the $CQCT_i^t$ (Chip Queue Complete Time) as the time to complete all the requests in the queue of chip i at time t , which is the waiting time of the next request to this chip. $CQCT_i^t$ covers the write/read costs, potential GC costs, and metadata costs. We find that, at most time, the $CQCTs$ of different chips vary in a wide range. To capture this, we define the unbalance rate²² of chip-level queues as

$$unbalance\ rate = \frac{\max_{1 \leq i \leq n} CQCT_i^t}{\sum_{i=1}^n CQCT_i^t / n},$$

where n is the number of chips in an SSD. In the above experiments, we also compute the cumulative distribution function of unbalance rate of chip-level queues during the runtime under LRU scheme, shown in Figure 4. We find that there are almost 60% of the cases that the unbalance rates are larger than 10, which means that the $CQCT$ of the busiest chip is more than 10 times bigger than the average. As a result, if we evict pages belonging to the chips with minimum $CQCT$, the delay of writing back operation will be reduced.

Based on the aforementioned experimental results and findings, we aim to design a novel cache replacement scheme, which takes into account the current states of waiting requests queues on chips, to improve the performance of SSDs.

3 | THE DESIGN OF ECR

Taking the eviction cost into consideration, we propose an Eviction-Cost aware Cache Replacement (ECR) algorithm in this paper to minimize the cost of replacing a dirty page and improve the request response time of SSDs. Figure 5 outlines the design of ECR, which is mainly composed of three key modules, namely, Cache Replacement Unit (CRU), Chip Queue Monitor Unit (CQMU), and Multi-LRU Structure (MLS). CRU chooses an

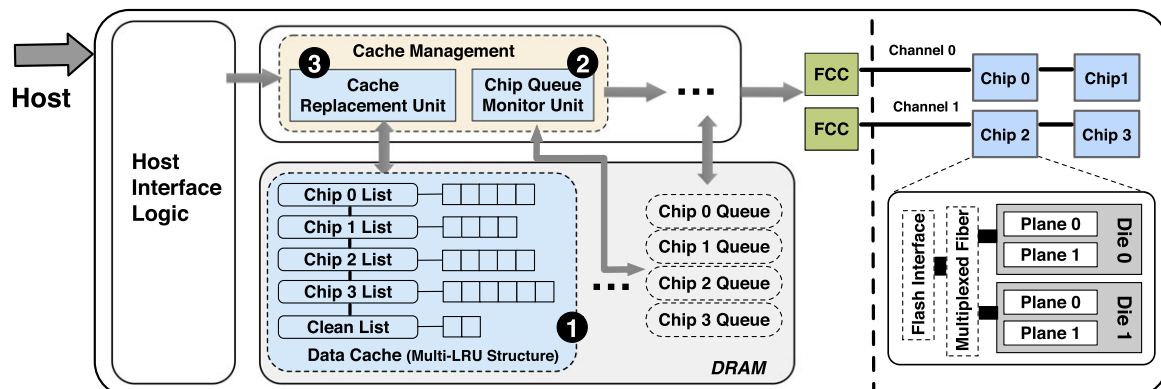


FIGURE 5 High-level overview of ECR

appropriate victim page to be evicted when cache is full. CQMU monitors the chip queues to provide necessary information for the decision of CRU. MLS organizes the pages in cache to support quick searches for pages.

The data structures of other replacement algorithms can be integrated into ECR easily. The conversion from the original cache algorithms to the ECR-based algorithms will be smooth, because ECR proposes a model to calculate the waiting time of different chips. In other words, the cached data in each chip can be managed by other algorithms, and just use CQMU module to select a target chip with shortest waiting time.

3.1 | Chip queue monitor unit

In SSDs, there is a chip-level queue for each chip to enqueue the requests accessing the data on it. Chip Queue Monitor Unit (CQMU) is used to compute the expected time for processing all the requests in a chip queue at current time, denoted as Chip Queue Complete Time (CQCT). CQCT also represents the waiting time of a new request arriving at this chip. The requests in a chip queue consist of write/read requests coming from users and imposed by Garbage Collection (GC). Therefore, CQCT covers the time of processing user's write/read requests and the time cost by GC and the time for writing the summary page if necessary. We formalize CQCT in Equation (1). The symbols in Equation (1) are explained in Table 1. In the SLC flash memory, the write/read latency is constant. However, in MLC and TLC, the write/read latencies of LSB page and MSB page may be different. In this equation, we assume that the write/read latency is constant to simplify the model, which makes ECR is only suitable for SLC flash memory. However, the basic idea is also suitable for MLC and TLC flash memories, and it just needs to adjust the latency according to the physical page, which it requests

$$CQCT_i = L_r * N_r^i + L_w * N_w^i + S_i * T_{gc}^i + L_{sp} * N_{sp}^i. \quad (1)$$

SSDs usually need to maintain a minimum number of free pages to perform data transfer during GC.² When the number of free pages is smaller than the predetermined threshold, the corresponding chip will activate GC process to recycle some invalid pages. The GC state can be formulated as Equation (2). GC process will first choose a victim block according to the GC scheme (eg, greedy scheme²³) to erase. Then, it will migrate the valid pages in this block to other free pages. A write and a read operation are needed for each valid page migration. At last, it will erase this block. Ultimately, the GC time cost is computed by Equation (3).

In a flash block, it usually uses the last flash page in this block to hold some summary information for this block. It means that, when an active flash block only has one free page left, it will write a summary page into this free page.²⁴ This summary page write operation is also time-consuming, whose latency is only a little smaller than the latency of a normal flash write. Therefore, we also take the summary page writes into consideration to compute CQCT, which can be derived as Equation (4)

$$S_i = \begin{cases} 0 & \text{if } N_{fp}^i - N_w^i > N_{mfp} \\ 1 & \text{if } N_{fp}^i - N_w^i \leq N_{mfp} \end{cases} \quad (2)$$

$$T_{gc}^i = L_{gc} + N_{vp}^i * (L_r + L_w) \quad (3)$$

$$N_{sp}^i = \begin{cases} 0 & \text{if } N_w^i + S_i * N_{vp}^i < N_{fpb}^i - 1 \\ 1 + (N_w^i + S_i * N_{vp}^i - (N_{fpb}^i - 1)) / (N_p^i - 1) & \text{if } N_w^i + S_i * N_{vp}^i \geq N_{fpb}^i - 1. \end{cases} \quad (4)$$

TABLE 1 A list of symbols

Symbol	Description
L_r	Latency of a flash read operation
L_w	Latency of a flash write operation
L_{gc}	Latency of a flash erase operation
L_{sp}	Latency of a summary page write
N_r^i	Number of read requests in Chip i Queue
N_w^i	Number of write requests in Chip i Queue
N_p	Number of pages in a flash block
N_{fp}^i	Number of free pages in Chip i at current time
N_{fpb}^i	Number of free pages in the active block in Chip i at current time
N_{mfp}^i	Minimum number of free pages in a chip, if the number of free page is less than this value, it will activate GC
N_{vp}^i	Number of valid pages in the victim block, which will be erased in Chip i
N_{sp}^i	Number of summary pages, which will be written in Chip i
S_i	A flag represents if Chip i will activate GC, 1 for Yes, 0 for No
T_{gc}^i	Time for a GC operation in Chip i

3.2 | Multi-LRU structure

In ECR, we use multi-LRU lists to organize the pages in cache rather than a single LRU list, where each LRU list contains dirty pages for each chip and a common *Clean-LRU* list contains the clean pages on all chips. If a read request misses in cache, it will turn to access this page on flash chip. The requested page will be loaded into cache, and marked as a clean page, inserted into the *Clean-LRU* list. If a write request hits a clean page in the *Clean-LRU* list, the page will be marked as a dirty page and will be moved from the *Clean-LRU* list to the LRU list of dirty pages in the corresponding chip according to its logical page number.

When a request arrives in cache management module, ECR first searches the data in *Clean-LRU* list. If it hits in *Clean-LRU* list, the requests can be serviced directly; otherwise, ECR gets the chip number from the request's address and then searches the data entry in the corresponding chip LRU list. The multi-LRU list structure accelerates the speed of the search operation in cache compared to a single LRU list.

Next, we consider the selection of victim pages for eviction and the impact of this selection on cache hit ratio. In ECR, we always firstly select a chip according to the information provided by Chip Queue Monitor Unit and then select the least recently used page belonging to this chip as a victim for eviction. Thus, multi-LRU lists will not bring any extra negative impact on cache hit ratio under the ECR scheme. In contrast, employing multi-LRU lists can not only accelerate the search operation in cache but also improve the speed of selecting a victim page for eviction.

3.3 | Cache replacement unit

Evicting a clean page in cache does not need a write back operation, so the eviction cost of a clean page is zero. ECR also prefers to evict a clean page in cache when cache is full like CFLRU.¹³ Keeping all the clean pages in a *Clean-LRU* list makes it faster to find a clean page in cache, especially when there are only a few clean pages in cache. In ECR scheme, when the cache is full, Cache Replacement Unit (CRU) firstly checks the *Clean-LRU* list. If it is not empty, CRU will choose the least recently used page in *Clean-LRU* list as the victim for eviction. Unfortunately, however, the *Clean-LRU* list is empty at most of the time for write-dominant applications. When the *Clean-LRU* list is empty, CRU will call Chip Queue Monitor Unit (CQMU) to compute the Chip Queue Complete Time (CQCT) for each chip. The CQCT of chip i represents how long it will wait to be executed, when a request arrives in chip i at current time. CQMU will always return a chip number with the minimum CQCT to CRU. Then, CRU will choose the least recently used dirty page in the LRU list of the chosen chip as a victim for eviction. To do so, ECR minimizes the waiting time of write back requests, and consequently minimizes the latency of eviction operation. Selecting the least recently used page in dirty pages LRU lists of chips introduces no negative impacts on the cache hit ratio. Because of multiple dirty page LRU lists and the common *Clean-LRU* list, ECR improves the performance of SSDs.

4 | PERFORMANCE EVALUATION

4.1 | System configuration

SSD Configurations. In our experiments, we use a trace-driven simulator, DiskSim²⁵ with SSD extension,² which has been widely used to evaluate the performance of SSDs. In this paper, we simulate an SSD of 64 GB with 64 chips, which implements a greedy garbage collection and dynamic wear leveling schemes. The SSD's over-provisioning ratio is set as 15%, the default setting of most SSDs.² The other parameters can be found in Table 2. In our experiments, we firstly warm up the SSD to reach a steady performance and then start to measure the performance of different schemes. The cache size is set to 32 MB unless otherwise specified. The other configurations and the FTL scheme are set to default in DiskSim. ECR is mainly optimized for the page-level mapping FTL as it shows a higher performance than block-level or hybrid-mapping FTL.¹ Therefore, we only conduct the experiments based on page-level mapping FTL.

Workloads. We first use six realistic enterprise scale workloads to evaluate the performance of different cache management schemes. ECR is mainly optimized for the write-dominant applications, so we collected some write-dominant traces from these block-level traces, which are collected from 36 volumes in an enterprise data center for one week.^{26,27} These traces come from a wide range of enterprise-scale application domains. The more information about these traces can be found in Table 3. At last, we also create a series of synthetic traces to study the efficiency of ECR under workloads with different write/read ratios and access patterns.

TABLE 2 SSD parameters²

Parameter	Value	Parameter	Value
SSD Capacity	64 GB	Page size	4 KB
# of chips	64	Page read Latency	25 us
# of planes per chips	8	Page write Latency	200 us
# of block per plane	512	Block erase Latency	1.5 ms
# of pages per block	64	Chip Xfer Latency(per byte)	0.025us

TABLE 3 Statistics of workloads

Trace	# of Requests	Write Ratio	Avg. Size (KB)	Description
hm_0	3 993 316	0.65	8.88	Hardware monitoring
prn_0	5 585 886	0.89	12.54	Print server
proj_0	4 224 524	0.88	38.11	Project directories
prxy_0	12 518 968	0.97	7.08	Firewall/web proxy
src1_2	1 907 773	0.75	29.26	Source control
src2_0	1 557 814	0.89	7.60	Source control

Baselines. In our evaluations, we compare ECR with three different cache management algorithms, LRU, CFLRU,¹³ and GCaR.¹⁷ LRU is the most widely used cache replacement algorithm. CFLRU and GCaR are more recent proposals summarized as follows.

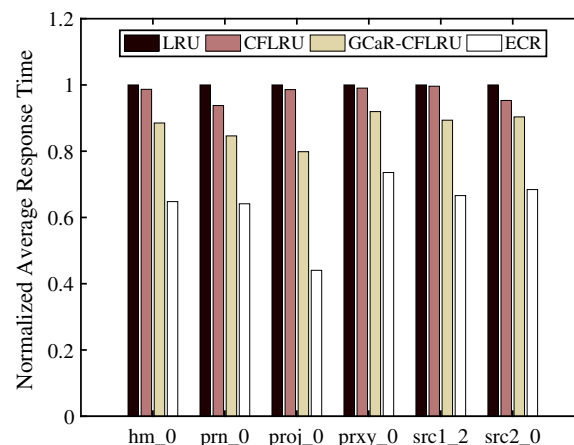
- **CFLRU** is a flash-aware cache management scheme. It splits LRU list into working region and clean-first region, and it prefers to evict clean pages in clean-first region. The performance of CFLRU is related to the size of clean-first region. Therefore, in our evaluation, we set the size of clean-first region as S/x , where S is the cache size and x varies from 1 to 6.
- **GCaR** can be regarded as the state-of-the-art scheme, which takes GC interference into consideration. It prefers to evict pages belonging to the chips not in the GC state. GCaR can be integrated into CFLRU and further improve the efficiency of the cache in SSDs. Therefore, we additionally compare the performance of ECR with GCaR-CFLRU scheme in our evaluations.

4.2 | Evaluations with realistic workloads

Response time. Figure 6 shows the normalized average response times of six traces under different cache management schemes. For these six traces, we find that ECR reduces the average response time by 26.33% to 59.55%, 25.69% to 55.32%, and 19.97% to 44.84% compared with LRU, CFLRU, and GCaR-CFLRU, respectively. The reasons for this improvement are as follows. There are only a few clean pages in cache for write-dominant traces; CFLRU can hardly find a clean page for eviction when cache is full, which makes CFLRU have the similar performance with LRU and only show a little improvement for some traces. Although GCaR-CFLRU can hardly benefit from the CFLRU scheme, it benefits from avoiding evicting dirty pages belonging to the chips in the GC state. Thus, GCaR-CFLRU achieves some improvement of the efficiency of page eviction compared to LRU. However, ECR aims to minimize the waiting time of writing back dirty pages, which minimizes the latency of the eviction operation and reduces the waiting time of the subsequent requests to the corresponding chips. As a result, ECR achieves largest improvement compared with other schemes. Especially for *proj_0*, ECR reduces the response time by 44.84% compared to GCaR-CFLRU. This is because *proj_0* is highly write-intensive and with highly skewed chip queues, which makes ECR more efficient.

Hit ratio. Figure 7 shows the cache hit ratios of six traces with different cache management schemes. We find that CFLRU, GCaR-CFLRU, and ECR has similar hit ratios with LRU. This is because these schemes are all based on LRU. Although they use some different techniques to improve LRU, they still use the LRU principle to utilize the access locality of workloads. We use the multi-LRU to organized the pages in cache, but we find that it only has a little negative impact on hit ratio, less than 10% for all six traces.

Standard Deviation of Response Time. Figure 8 shows the normalized standard deviation of response times of six traces with different cache management schemes. For these six traces, we find that ECR reduces the standard deviation of response time by 25.50% to 63.46%, 21.77% to 63.04%, and 14.61% to 53.18% compared to LRU, CFLRU, and GCaR-CFLRU, respectively. ECR always tries to evict the pages belonging to

**FIGURE 6** Average response time

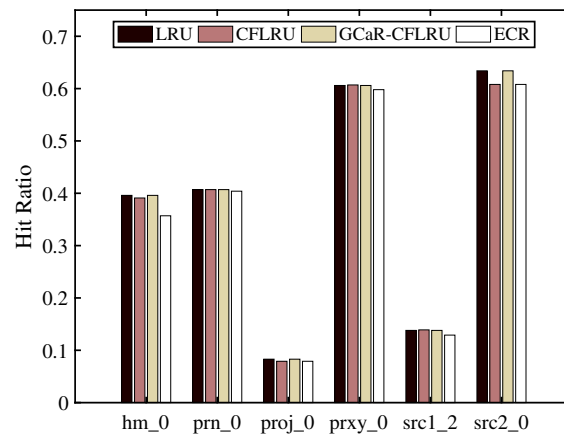


FIGURE 7 Cache hit ratio

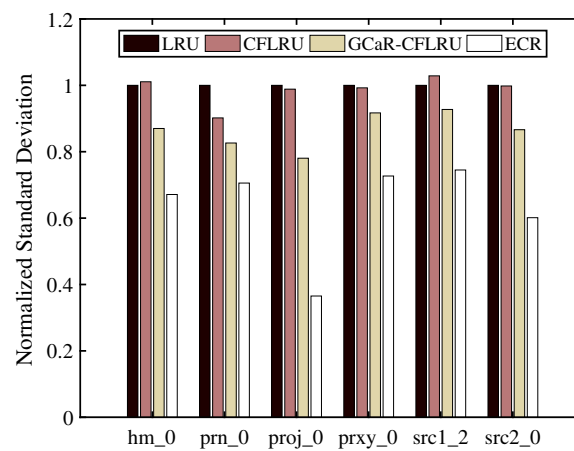


FIGURE 8 Std. of response time

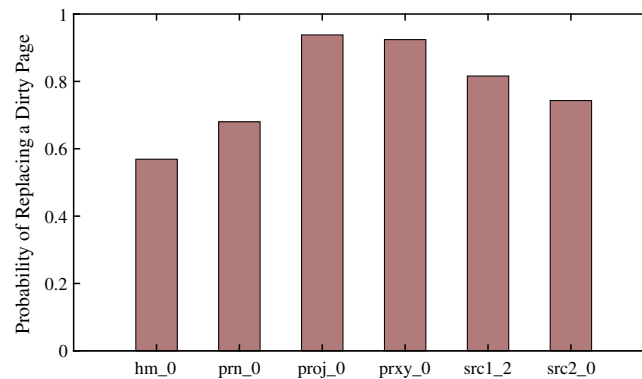


FIGURE 9 Probability of replacing a dirty page in ECR

the chips with the smallest chip queue complete time, which alleviates the unbalanced time cost for different chips to complete the requests on them and achieves great improvements on standard deviation of response time.

Insights of ECR. ECR benefits from not only the preference to evicting pages in *Clean-LRU* list, but also minimizing eviction cost. Figure 9 shows the probability of replacing dirty pages with ECR. We find that it has a very high probability of replacing dirty pages with ECR for all six traces, which means that preference to evicting clean pages in *Clean-LRU* list only contributes a little to the improvement of ECR. Furthermore, we give detailed evaluations about ECR. First, we only prefer to evict clean pages in *Clean-LRU* list and turn to the least recently used page in the whole LRU list if there is no clean page. This scheme is very like CFLRU⁵, just setting the size of clean-first region being equivalent to the cache size, so we name this scheme as CFLRU⁶. Then, we ignore the *Clean-LRU* list in ECR, all dirty pages, and clean pages being organized in the corresponding chip LRU list. We name this scheme as ECR⁷. It always evicts the least recently used page in the chip LRU list whose chip queue complete time is the minimum. Figure 10 shows the normalized average response times of these schemes. We find that CFLRU⁶ only has a little improvement

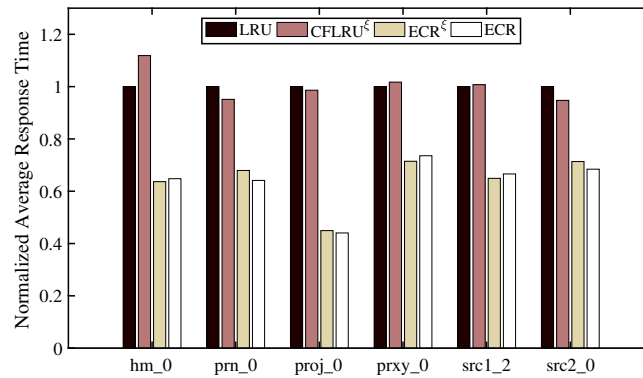


FIGURE 10 Normalized average response time

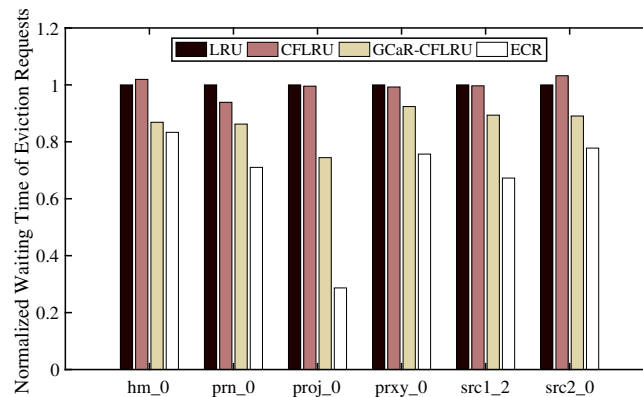


FIGURE 11 Normalized waiting time of eviction requests

TABLE 4 Comparisons of waiting time reduction and response time reduction

Trace	Reduction of waiting time	Reduction of response time	Trace	Reduction of waiting time	Reduction of response time
hm_0	0.1667	0.3522	prxy_0	0.2432	0.2644
pm_0	0.2898	0.3587	src1_2	0.3273	0.3340
proj_0	0.7133	0.5596	src2_0	0.2221	0.3158

compared to LRU for most of the traces, even has a poorer performance than LRU for *hm_0* because of the negative impact on hit ratio. These results lead to the conclusion that evicting clean pages in ECR may only contribute a little to its improvement. The performance of ECR^ε is very close to that of ECR, as shown in Figure 10, even that ECR^ε shows slightly shorter response time than ECR for *hm_0* because of the bad performance of CFLRU^ε. Although both of preference to evicting clean pages and being aware of eviction cost contribute to ECR, most of the improvement of ECR comes from being aware of eviction cost.

Waiting time of eviction requests. ECR always prefers to evict a dirty page belonging to the chip which needs shortest time to complete all requests in its queue. This can make eviction write requests have the shortest waiting time. Figure 11 shows the normalized waiting times of eviction requests under different schemes. They are normalized to that of LRU schemes. We can find that ECR can reduce the waiting time of eviction requests by 13.14% to 71.33%, 18.70% to 71.20%, and 4.25%-61.48%, compared to LRU, CFLRU, and GCaR-CFLRU, respectively. The reduction of waiting time makes the eviction requests be completed quickly, which can reduce the waiting times of normal requests. Therefore, ECR can reduce the response time of user requests. We compare ECR with LRU scheme, and we show the reduction of waiting time of eviction operations and the reduction of response time in Table 4. We can find that the more reduction of the waiting time is, the more reduction of the response time is. *proj_0* has a great reduction on waiting time of eviction operations, reaching to 71.33%, and accordingly, the reduction of response time reaches to 55.96%.

Response time under background GC. Figure 12 shows the normalized average response times under different schemes employing background garbage collection. We can find that ECR can also reduce the response time by 25.92% to 56.09%, 25.52% to 55.66%, and 20.10% to 45.38% compared to LRU, CFLRU, and ECaR-CFLRU, respectively. Although it can process GC in background, the GC operation will also compete with normal write/read requests in write-intensive workloads. If one of write requests in the chip queue is written to the last free page, background GC will be evoked, and then the subsequent requests in the chip queue must wait to be handled until the GC process is completed. If the write requests are not intensive, after a write request is written to the last free page, it has enough idle time to process GC before the next write

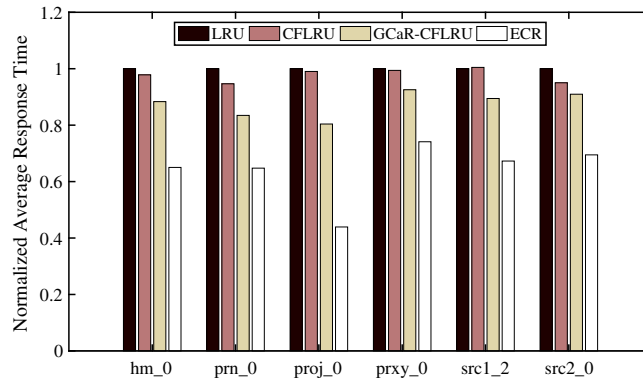


FIGURE 12 Normalized average response time

request arrives. In this situation, ECR may only have little improvement. Therefore, we can conclude that ECR also can make great improvement in write-intensive workloads although it employs background GC.

4.3 | Evaluation with synthetic workloads

To evaluate the performance of ECR for different types of workloads, we use some synthetic workloads to evaluate ECR. We use both uniformly and normally distributed workloads and vary the write ratio of these two types of workloads from 20% to 80%. In our experiments, the logical page numbers in uniform distributed workload are uniformly distributed in a 300 MB range, and the standard deviation of the logical page numbers in normal distributed workload is set to 10 000. The average request sizes of all workloads are equal to the size of a flash page, and the numbers of requests in all workloads are 2 000 000.

Uniformly Distributed Workloads. Figure 13 shows the normalized average response times with different cache management schemes for the uniformly distributed workloads with different write/read ratios. We find that, when the write ratio is 20%, CFLRU, GCaR-CLRU, and ECR only have a little improvement compared to LRU scheme. This is because there are only a few write requests, which only introduces a limited number of dirty pages in cache and a few GC operations. In this case, as all baselines including CFLRU, GCaR-CFLRU, and ECR give a high probability to select a clean page for eviction, with the increasing write ratios, ECR significantly outperforms those schemes, and the improvement reaches to the maximum when the write ratio is 80%. This is because there are a few clean pages in cache when the write ratio is 80%, and the victim page has a very high probability of being a dirty page. Moreover, ECR achieves better performance by carefully selecting the evicted pages to minimize the costs of writing back dirty pages. CFLRU performs slightly better than LRU. GCaR-CFLRU also achieves better performance with the increasing write ratios, compared to all schemes except ERC. This is because GCaR-CFLRU only takes the current GC state into consideration. Figure 14 shows the hit ratios with different cache management schemes for the uniformly distributed workloads with different write/read ratios. We find that all the schemes have the same cache hit ratios. Because the accessed pages are uniformly distributed, there is no much temporal locality that can be utilized, which makes cache hit ratio be constant no matter which cache management scheme we use.

Normally Distributed Workloads. Figure 15 shows the normalized average response time with different cache management schemes for the normally distributed workloads with different write/read ratios. When the write ratio becomes larger, there are more dirty pages in cache so that in all schemes dirty pages have a higher probability to be evicted and more write back operations are introduced. ECR aims to select an appropriate dirty page to minimize the cost of write back operation, while the other schemes are not aware of that cost. Thus, when the write

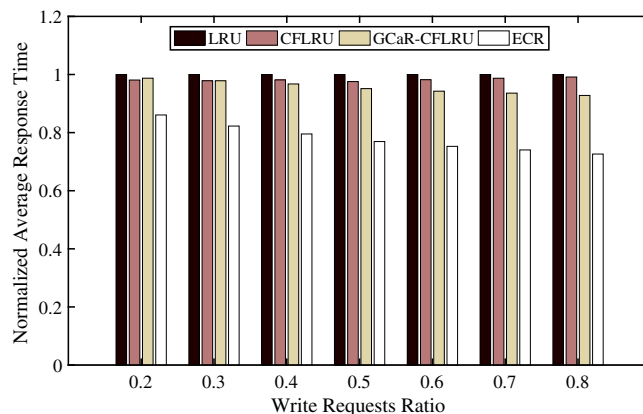


FIGURE 13 Normalized response times of uniform distributed workloads

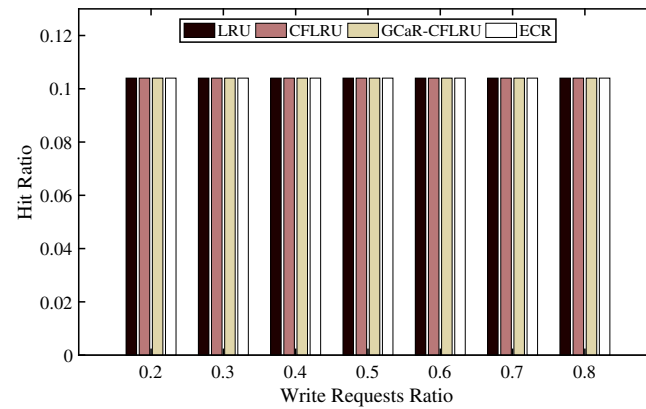


FIGURE 14 Hit ratios of uniform distributed workloads

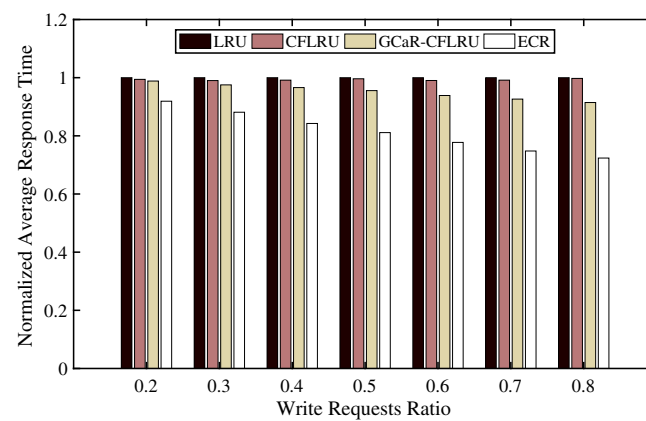


FIGURE 15 Normalized response times of normal distributed workloads

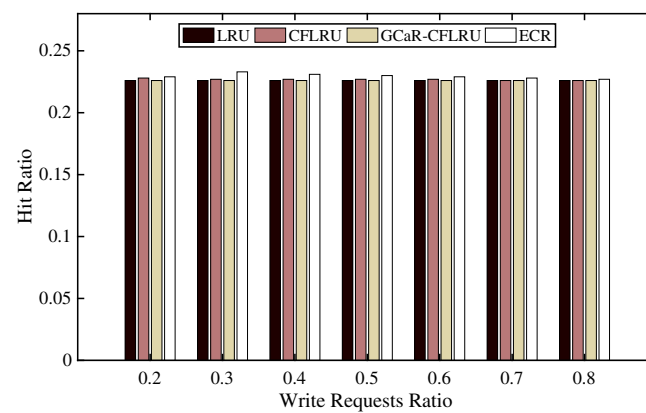


FIGURE 16 Hit ratios of normal distributed workloads

ratio reaches about 0.7 or higher, ECR performs significantly better than other schemes. Figure 16 shows the hit ratios with different cache management schemes. Although the workloads are normal distributed, we find that ECR still has the similar hit ratio to LRU because of the careful selection of the victim page.

5 | RELATED WORKS

Cache is one of the most important design to improve the performance of storage systems.²⁸ For cache replacement, there have been plenty of studies. Least Recently Used (LRU) is the most common used algorithm for cache management. There are also many algorithms based on LRU

and extending it for some particular workloads, such as 2Q,⁸ LRU-K,⁹ ARC,¹⁰ and LIRS.¹¹ However, all these algorithms only pay attention to improve the cache hit ratio.¹³ Thus, they greatly improve the performance of HDDs. However, this improvement cannot be directly applied for flash-based SSDs¹² due to the unique characteristics in SSDs.

There are also some flash-aware cache policies being proposed. CFLRU¹³ assigns a higher priority to dirty pages to stay in cache. It splits the LRU list into the working region and the clean-first region. It prefers to evict the clean pages in the clean-first region, and turns to evict the least recently used page if there is no clean page in clean-first region. However, there are many dirty pages in cache for write-dominant applications. Therefore, CFLRU can hardly find the clean pages in clean-first region and often evict the least recently used page just like LRU algorithm. There are also some schemes been proposed to optimize CFLRU. LRU-WSR¹⁴ pays more attention to reduce the number of write operations. It adds an additional cold flag to mark the cold pages and tries not to keep cold dirty page in cache. It prefers to select a clean page as a victim without checking the status of the cold flag. CCF-LRU¹⁵ takes the access frequency of clean pages into consideration, and it classifies the clean pages into cold and hot ones. It prefers to evict cold clean pages and the eviction of hot clean pages will be delayed. AD-LRU¹⁶ maintains two LRU lists, ie, cold LRU list and hot LRU list, where cold LRU list stores the pages that only accessed once and hot LRU list stores the pages that accessed at least twice. However, these algorithms are all based on CFLRU, and they face the same problem as CFLRU. They can hardly find clean pages for eviction in write-dominant applications and are not aware of the cost when evict dirty pages. BPAC,²⁹ LB-CLOCK,³⁰ PUD-LRU,³¹ and REF³² are all proposed to improve the cache management base on block-level or hybrid mapping FTL. However, in this paper, we only consider the page-level mapping FTL as it shows a higher performance than block-level or hybrid-mapping FTL.¹ There are also some works³³⁻³⁶ that take the relationships between cache management policies and some other FTL functions into consideration.

More recently, GCaR¹⁷ takes the state of garbage collection into consideration. GCaR assigns higher priority to the pages belonging to flash chips that are in GC state to stay in cache. Therefore, GCaR reduces the contentions between the I/O caused by eviction and the I/O caused by GC. However, it does not consider the queuing states of different chips, which may evict dirty pages to the chips with longer waiting queue. Our ECR evicts dirty pages belonging to the chips with the shortest waiting time, which speeds up the write back operation of dirty pages and further improves the performance of cache replacement.

6 | CONCLUSIONS

The traditional cache replacement algorithms mostly focus on improving hit ratio or trying to evict clean pages. They are not aware of eviction cost. In an SSD, there are a number of chips with skewed queues on different chips. Therefore, the cost of write back operations for evicting dirty pages may be varied in a wide range. Taking this into consideration, we have proposed an eviction-cost aware cache replacement algorithm to minimize the time cost of evicting a dirty page. We give a higher probability to evict pages belonging to the chips with the shortest time to complete the pending requests in their chip queues. To quickly select a victim page in cache, we design a multi-LRU list to organize the pages in cache. Our experimental results show that ECR significantly reduces the average response time compared to state-of-the-art cache replacement algorithms.

ACKNOWLEDGMENT

This work was supported by National Nature Science Foundation of China under Grant No. 61772486. The work of Yubiao Pan was supported by National Nature Science Foundation of China under Grant 61802133 and Nature Science Foundation of Fujian Province under Grant 2018J05107.

ORCID

Hao Chen  <https://orcid.org/0000-0001-8249-7046>

REFERENCES

1. Gupta A, Kim Y, Urgaonkar B. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In: Soffa ML, Irwin MJ, eds. *ASPLOS XIV: Fourteenth International Conference on Architectural Support for Programming Languages and Operating Systems, March 7-11, 2009, Washington, DC, USA*. New York, NY: ACM; 2009:229-240.
2. Agrawal N, Prabhakaran V, Wobber T, Davis JD, Manasse MS, Panigrahy R. Design tradeoffs for SSD performance. In: *Proceedings of the USENIX Annual Technical Conference (ATC'08)*; 2008; Boston, MA.
3. Liu C, Lv M, Pan Y, et al. LCR: load-aware cache replacement algorithm for flash-based SSDs. In: *Proceedings of the 2018 IEEE International Conference on Networking, Architecture and Storage (NAS)*; 2018; Chongqing, China.
4. Shim H, Seo B-K, Kim J-S, Maeng S. An adaptive partitioning scheme for DRAM-based cache in solid state drives. In: *Proceedings of the IEEE 26th Symposium On Mass Storage Systems and Technologies (MSST)*; 2010; Incline Village, NV.
5. Selvan Ramasamy A, Karantharaj P. RFLRU: a buffer cache management algorithm for solid state drive to improve the write performance on mixed workload. *Engineering Letters*. 2014;22(4).
6. Jain A, Lin C. Back to the future: leveraging Belady's algorithm for improved cache replacement. In: *Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*; 2016; Seoul, South Korea.
7. Ding X, Jiang S, Chen F. A buffer cache management scheme exploiting both temporal and spatial localities. *ACM Trans Storage*. 2007;3(2). Article No. 5.

8. Johnson T, Shasha D. 2Q: a low overhead high performance buffer management replacement algorithm. In: Bocca JB, Jarke M, Zaniolo C, eds. *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB 94)*. Burlington, MA: Morgan Kaufmann Publishers; 1994:439-450.
9. O'neil EJ, O'neil PE, Weikum G. The LRU-K page replacement algorithm for database disk buffering. *ACM SIGMOD Rec.* 1993;22(2):297-306.
10. Megiddo N, Modha DS. ARC: a self-tuning, low overhead replacement cache. In: *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST '03)*; 2003; San Francisco, CA.
11. Jiang S, Zhang X. LIRS: an efficient low inter-reference recency set replacement policy to improve buffer cache performance. *ACM SIGMETRICS Perform Eval Rev.* 2002;30(1):31-42.
12. Kim H, Ahn S. BPLRU: A buffer management scheme for improving random writes in flash storage. In: Baker M, Riedel E, eds. *USENIX Conference on File and Storage Technologies (FAST 2008)*. Berkeley, CA: USENIX Association; 2008:239-252.
13. Park S, Jung D, Kang J, Kim J, Lee J. CFLRU: a replacement algorithm for flash memory. In: *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*; 2006; Seoul, South Korea.
14. Jung H, Shim H, Park S, Kang S, Cha J. LRU-WSR: integration of LRU and writes sequence reordering for flash memory. *IEEE Trans Consumer Electron.* 2008;54(3):1215-1223.
15. Li Z, Jin P, Su X, Cui K, Yue L. CCF-LRU: a new buffer replacement algorithm for flash memory. *IEEE Trans Consumer Electron.* 2009;55(3):1351-1359.
16. Jin P, Ou Y, Härder T, Li Z. AD-LRU: an efficient buffer replacement algorithm for flash-based databases. *Data Knowl Eng.* 2012;72:83-102.
17. Wu S, Lin Y, Mao B, Jiang H. GCaR: garbage collection aware cache management with improved performance for flash-based SSDs. In: *Proceedings of the 2016 International Conference on Supercomputing*; 2016; Istanbul, Turkey.
18. Margaglia F, Yadgar G, Yaakobi E, Li Y, Schuster A, Brinkmann A. The devil is in the details: implementing flash page reuse with WOM codes. In: *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST)*; 2016; Santa Clara, CA.
19. Elyasi N, Arjomand M, Sivasubramaniam A, Kandemir MT, Das CR, Jung M. Exploiting intra-request slack to improve SSD performance. *ACM SIGOPS Oper Syst Rev.* 2017;51(2):375-388.
20. Tavakkol A, Gómez-Luna J, Sadrosadati M, Ghose S, Mutlu O. MQSim: a framework for enabling realistic studies of modern multi-queue SSD devices. In: *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST)*; 2018; Oakland, CA.
21. Gao C, Shi L, Zhao M, Xue CJ, Wu K, Sha EH-M. Exploiting parallelism in i/o scheduling for access conflict minimization in flash-based solid state drives. In: *Proceedings of the Symposium on Mass Storage Systems and Technologies (MSST)*; 2014; Santa Clara, CA.
22. Shen Z, Shu J, Lee PPC. Reconsidering single failure recovery in clustered file systems. In: *Proceedings of the 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*; 2016; Toulouse, France.
23. Bux W, Iliadis I. Performance of greedy garbage collection in flash-based solid-state drives. *Performance Evaluation.* 2010;67(11):1172-1186.
24. Birrell A, Isard M, Thacker C, Wobber T. A design for high-performance flash disks. *ACM SIGOPS Oper Syst Rev.* 2007;41(2):88-93.
25. Bucy JS, Schindler J, Schlosser SW, Ganger GR. *The DiskSim Simulation Environment Version 4.0 Reference Manual (CMU-PDL-08-101)* [technical report]. Pittsburgh, PA: Parallel Data Laboratory, Carnegie Mellon University; 2008:26.
26. Narayanan D, Donnelly A, Rowstron A. Write off-loading: practical power management for enterprise storage. *ACM Trans Storage.* 2008;4(3). Article No. 10.
27. Microsoft Research. <http://iota.snia.org>
28. Hsu WW, Smith AJ. The performance impact of I/O optimizations and disk improvements. *IBM J Res Dev.* 2004;48(2):255-289.
29. Wu G, Eckart B, He X. BPAC: an adaptive write buffer management scheme for flash-based solid state drives. In: *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*; 2010; Incline Village, NV.
30. Debnath B, Subramanya S, Du D, Lilja DJ. Large Block CLOCK (LB-CLOCK): a write caching algorithm for solid state disks. In: *Proceedings of the 2009 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems*; 2009; London, UK.
31. Hu J, Jiang H, Tian L, Xu L. PUD-LRU: an erase-efficient write buffer management algorithm for flash memory SSD. In: *Proceedings of the 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*; 2010; Miami Beach, FL.
32. Seo D, Shin D. Recently-evicted-first buffer replacement policy for flash storage devices. *IEEE Trans Consumer Electron.* 2008;54(3):1228-1235.
33. Liao XL, Hu SM. Bridging the information gap between buffer and flash translation layer for flash memory. *IEEE Trans Consumer Electron.* 2011;57(4):1765-1773.
34. Boukhobza J, Olivier P, Rubini S, Lemarchand L, Hadjadj-Aoul Y, Laga A. MaCACH: an adaptive cache-aware hybrid FTL mapping scheme using feedback control for efficient page-mapped space management. *J Syst Archit.* 2015;61(3-4):157-171.
35. Boukhobza J, Olivier P, Rubini S. A cache management strategy to replace wear leveling techniques for embedded flash memory. In: *Proceedings of the 2011 International Symposium on Performance Evaluation of Computer & Telecommunication Systems*; 2011; The Hague, The Netherlands.
36. Park S, Cha J, Kang S. Integrated write buffer management for solid state drives. *J Syst Archit.* 2014;60(4):329-344.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of the article.

How to cite this article: Chen H, Pan Y, Li C, Xu Y. ECR: Eviction-cost-aware cache management policy for page-level flash-based SSDs. *Concurrency Computat Pract Exper.* 2019;e5395. <https://doi.org/10.1002/cpe.5395>