LCache: Machine Learning-Enabled Cache Management in Near-Data Processing-Based Solid-State Disks

Hui Sun, Shangshang Dai, Qiao Cui, Jianzhong Huang

LCache: Machine Learning-Enabled Cache Management in Near-Data Processing-based Solid-State Disks*

Hui Sun[1][0000−1111−2222−3333], Shangshang Dai[0000−0002−0721−8649]1, Qiao Cui[0000−0003−0415−5184]1, and Jianzhong Huang[2][2222−−3333−4444−5555]

[1] Anhui University, Hefei, Anhui, China, 230601
[2] Huazhong University of Science and Technology, Wuhan, Hubei, China, 430074
sunhui@ahu.edu.cn shangshangdai@stu.ahu.edu.cn qwcuiqiao@qq.com
hjzh@hust.edu.cn(corresponding author)

Abstract. In the era of big-data, large-scale storage systems use NAND Flash-based solid-state disks (SSDs). Some upper-level applications put higher requirements on the performance of SSD-based storage systems. SSDs typically exploit a small amount of DRAM as device side cache, yet the limitation of the DRAM inside an SSD makes a better performance difficult to achieve. The wide application of the existing cache management schemes (e.g., LRU, CFLRU) provides a solution to this problem. With the popularity of near-data processing paradigm in storage systems, the near-data processing-based SSDs are designed to improve the performance of the overall system. In this work, a new cache management strategy named LCache is proposed based on NDP-enabled SSD using a machine learning algorithm. LCache determines whether I/O requests will be accessed in a period by trained machine learning model (e.g., decision tree algorithm model) based on characteristics of I/O requests. When the infrequently accessed I/Os that are not intensive are directly flushed into the flash memory, LCahe enables to update the dirty data that has not been accessed in the cache to the flash memory. Thus, LCache can generate clean data while replace cached data with priority to minimize the cost of data evicting. LCache also can effectively achieve the threefold benefits: (1) reducing the data access frequency to frequently access data pages in flash memory, (2) improving the response time by 59.8%, 60% and 14.81% compared with LRU, CFLRU, and MQSim, respectively, and (3) optimizing the performance cliff by 68.2%, 68%, and 30.2%, respectively.

Keywords: Near-Data Processing · Machine Learning · Cache Management Solid State Disks.

* This work is supported in part by the Natural Science Research Projects at Higher Institutions in Anhui Province (KJ2017A015), National Natural Science Foundation of China under Grants 61702004, 61572209, and University Synergy Innovation Program of Anhui Province under Grants GXXT-2019-007.

1 Introduction

NAND Flash based solid state disks (SSDs) usually use a small amount of DRAM as device-side cache, which services I/O requests in high-performance DRAM, to effectively improve the storage performance of SSDs-based systems. However, the cache size is usually minimal because of its limited space and high cost. Academia and industry have been devoted to design cache management with high space utilization, durable robustness, and high performance. When the cache space is full, the items in cache must be wiped out from the cache. The existing practice of cache management usually use statistical methods. Data pages that are rarely used recently are removed from the cache. The cache eviction is triggered when a new data is written into the cache space which is full. Thus, in this case, there are two issues affecting the overall cache performance:

1) These evicted data may enter into the cache again due to the temporal locality and spatial locality that the access patterns exhibit.
2) The data update in cache is greatly affected by I/O access pattern. When I/O access is intensive, it is easier to replace cache. However if the bandwidth of flash memory is occupied, rendering confliction for I/O access. As such, even if the cache is full, cache replacement is less likely to be triggered, and the amount of SSDs bandwidth remains idle.

Machine learning is used in computing and storage systems[1]. With the improvement of computing capability of devices, an architecture called near-data processing (i.e., NDP) [2] or in storage computing (i.e., ISC) [3] attracts attention from scholars and professionals. In NDP devices, extra computing resources are usually used to provide higher computing power. Therefore, to tackle the issues, we propose a novel cache management algorithm, named LCache, based on the NDP-enabled SSDs with adopting machine learning on the computing resources in the NDP disk. LCache predicts that the data pages marked with the tags of clean, dirty, recency, and frequency will be wrote into the cache until they are replaced according to the characteristics of applications and then proactive update method is designed for flushing the data into the flash memory, meanwhile, deleting the data page in the cache.

When user data is not accessed by the machine learning, LCache judges the status of the page in flash memory, for example if the flash chip is idle, the dirty and recency cached item is updated into the flash memory, which can balance bandwidth utilization in the cache. The clean data has the priority to be replaced to minimize the cost of replacement.

The reason that the machine learning model-based LCache is applied to the NDP devices is that the LCache needs computing resources to process prediction based on I/Os characteristics. The traditional processor inside SSDs is difficult to provide corresponding computing power. LCache can handle the problem of the computing resources during the model running in NDP device. The machine learning model of LCache is deployed into FTL through firmware.

Contributions of this paper are summarized as:

1) In this paper, we design a cache management, named LCache, for NDP-based SSDs. LCache employs the decision tree C 4.5-based model to determine whether I/Os requests will access the cache in a while.
2) In LCache, we design a proactive update mechanism, by which the machine learning model is able to judge the data pages that will not be accessed again in the future. These pages identified with dirty-recency are flushed into the flash memory when the flash chips are idle. This method can balance the bandwidth cost in flash memory. Clean data is the first replaced to improve the overall performance of cache management.
3) We evaluate the performance of LCache in real-world enterprise traces. Experimental results show that the performance of LCache in terms of response time and performance cliff. LCache reduces the response time by 59.8%, 60%, and 14.81%, respectively, compared with LRU, CFLRU and MQSim under proj_0. The performance cliff is optimized by 68.2%, 68%, and 30.2%, respectively.

2 Background and Related work

2.1 Near-Data Processing Storage Device

Fig. 1 shows the overview of NDP storage system. The host connects with the NDP device through the PCIe NVMe protocol. The NDP device side is a flash-based solid-state disk with higher computing power. The difference between the NDP device and the typical solid-state disk is that the former has a hardware acceleration computing unit. The NDP device-side mainly consists of an embedded processor, hardware acceleration processor, and DRAM. The physical structure of flash memory is primarily divided into channels, chips, dies, planes, etc. Each block contains several pages. The flash memory chip can perform write, read, and erase operations. During this operation, the flash memory chip will block other access commands, making the blocked commands wait in the corresponding chip queue.

2.2 Cache Management

There are many cache management methods proposed in the past, such as LRU[4]. However, in SSDs, the penalty of cache miss resulting in cache replacement is different. Park et al. proposed the CFLRU [5], in which the cost of cache replacement could be reduced by replacing the consistent data in the cache and flash memory. With the popularity of machine learning, Wang et al. used machine learning to predict data access patterns in the cache, which could help to improve the cache strategy of "one-time access exclusion" [1]. In Tencent QQ application, the HDDs in Tencent storage servers are used as the back-end storage and SSD is used as the cache. They use the machine learning to cache the photos that will be accessed in the cache and these image data that will not be located will not be cached, by which the cache space is able to serve more

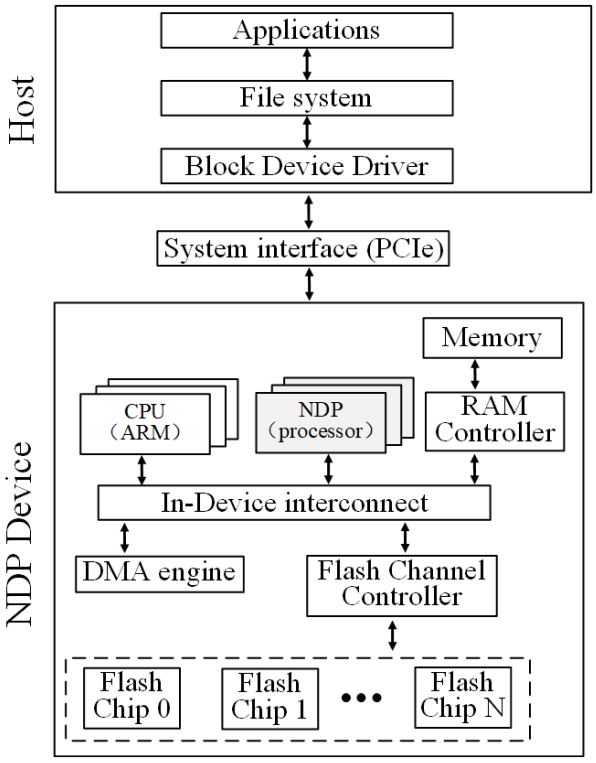

Fig. 1. Overview of Near-Data Processing-based SSDs.

applications, thereby optimizing the SSD lifetime. MQSim [6] enclosed a new caching scheme based on the write cache inside SSD. MQSim combined hot and cold in the write-through mode, in which cold data written in the cache would be flushed to the back-end flash memory. Meanwhile, the cached data in the flash memory would be promptly deleted, thereby keeping the cache release space. The following data can be directly written into the cache instead of the eviction operation, thus, dramatically reducing the frequency of cache replacement and providing higher performance. However, there is a large amount of data written into the flash memory, resulting in a large number of erasures.

3 Overview system

LCache is an active updating cache management strategy based on machine learning with the feature of low cost, high performance and life-friendly for NDP devices. The machine learning model is trained according to different enterprise application load characteristics. By deploying the trained machine learning model in the FTL layer of NDP devices, LCache, which adds functions to make it configurable, adequately judges whether I/O requests will re-access the cache. Simultaneously the machine learning model can decide whether the data pages will re-access the flash memory or not. When the flash memory chip is idle, these cached data pages are actively flushed into the flash memory, releasing the corresponding cache space. Even if the cache space is full, the cache will not be able to be accessed. To improve the overhead of the cache replacement, the active update mechanism is used to update recency dirty data into the flash memory when the flash memory chip is idle and then, the dirty data converts to clean one.

Fig. 2 presents the system structure of LCache. The NDP device connects with the host through the PCIe NVMe protocol. The host interface logic (HIL) is responsible for splitting I/O requests from the host into several flash transactions according to the size of the flash page. Flash transaction is the basic unit of cache entries and the transaction schedule unit (TSU). We use an accelerator inside the NDP device to extract features of the machine learning trained I/O requests. The machine learning model is suitable for enterprise application.

4 System Implementation

4.1 Machine Learning Module

To predict whether the cache data will be accessed again, we use decision tree C4.5 to train the machine learning model with the real enterprise application load. The data set is selected from UMass trace [7] and Microsoft Cambridge Research [8] with a wide range of enterprise applications. The decision tree algorithm is selected for the following reasons:

1) The decision tree is one of the typical machine learning algorithms. Its computational cost is relatively low. The size of the trained model is small, about

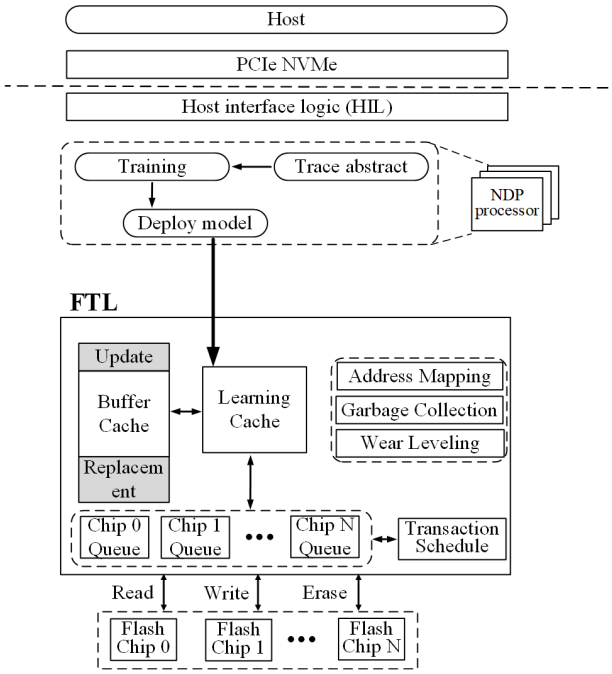

Fig. 2. The System Structure of LCache in NDP-based SSDs.

200 bytes. The cost of the machine learning algorithm is high in storage devices with less abundant resources.

2) The model trained by the decision tree algorithm can be easily transformed into an if-else statement and implemented in C language on the device side pursuing performance.
3) The accuracy of the trained model in our test can reach 80% - 90%, which can meet our application requirements.

We describe the generation pipeline of the machine learning model with three stages, i.e., feature extraction, training set generation and model training.

Feature extraction. A block-level I/O request contains time, device, LSN, size, operation type, etc. The feature extraction stage first splits an I/O request by the page size and then counts the frequency, distance, and size of LPN in a specific range in the past. Frequency refers to the number of times in the past to access the LPN within the cache size range. Distance is the number of separated pages of the same LPN to the current nearest access LPN in the past cache size. If there is no number found, the cache size minus 1. The revisited distance means finding the closest page that is the same as the current LPN and calculating the number of pages separated from the current page without limiting the cache size. If it exceeds the current preset range, it is set to -1.

Training set generation. When the features of each I/O request are extracted, the situation of entering and exiting the cache will be simulated accord-

ing to the LRU. If all tags of pages in an I/O request miss, the tag of this trace is missing. Except this scenario, we consider others shall be hit.

Model training. The training method adopted is decision tree C4.5, which is characterized by frequency, distance, and the size of a trace. The labels are 0 and 1, where "0" indicates that it will not be hit during the cache period. The notation "1" means that it will be hit. The trained model is able to classify data. According to the characteristics of the data in workload, it can be divided into (1) data will be hit in the cache (2) data will not be hit during the cache.

4.2 Proactive Update Function

The proactive update function is an essential component inside LCache. Two types of LRU linked lists (e.g., clean LRU link list and dirty LRU link list) are used to manage data in the write-buffer cache. This method can effectively search data and process the dirty or clean data to release cache space for storing the data pages that will be re-accessed in the future. Learning cache uses the update and replacement mechanism to reduce the cost of cache replacement.

Update Scheme. The cache management usually uses the write-back mechanism to ensure the data consistency between cache and flash memory. However, this mechanism is largely dependent on the I/Os access in applications, which quickly causes an imbalance of data flow from the cache to flash memory. When the I/O accesses is intensive, data updating from the cache to flash memory is frequent, which is prone to cause I/O access conflict.

As shown in Fig. 3, LCache triggers actively update judgment once new data is inserted into the cache. Then, it judges the data item that meets the requirements of active updates from the tail end of the dirty LRU list. LCache will check first if the chip is idle. When the flash memory chip corresponding to the LPA of the cache entry is idle and no flash transaction is waiting for service in the corresponding chip queue, the LPA corresponding chip will be identified as idle. If the other way around, the cache entry will not trigger an active update but proceed to check the next cache entry. LCache will execute Classifier using machine learning model. As shown in Figure 3, Classifier outputs 0 and 1. "1" represents that the data page is likely to be accessed again in the future. "0" means that the data page is unlikely to be accessed.

When the output of Classifier is 0, the data will be actively updated and written into the flash memory. The corresponding cache entries in the write buffer cache will be deleted as the data is unlikely to be accessed again in the future, thereby freeing the space for other cache items that may be accessed.

When the output of the Classifier is 1, LCache first judges whether there is space in clean data LRU list due to it accounts for much smaller proportion in cache space than that of the dirty-data LRU linked list. If the clean data LRU list is not full, it will be separated by hot and cold via distinguishing the data from the occurrence or frequency. This method can improve SSD lifetime caused by excessive active updating. A Bloom filter is used to record the coldness and hotness of each request, similar to that in Hsieh's work [9]. According to the spatial and temporal locality, the probability of reaccessing the recency data is

small. LCache actively updates the recency data into flash memory and move the cached dirty data LRU list to the clean data LRU list. If the differentiated data is frequency, then it will not be updated actively, and LCache continues to assess whether the next data item in the linked list will trigger the active update. Actively updating the recency data to flash memory and putting it into the clean data LRU list can minimize the cache replacement cost.

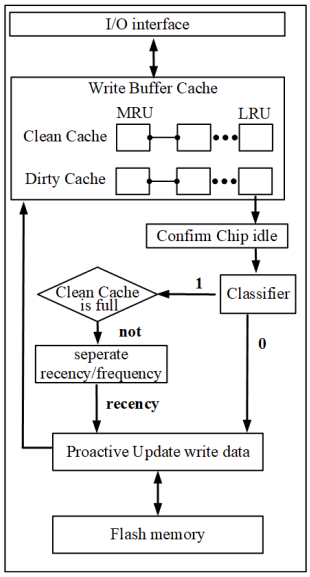

Fig. 3. The workflow in LCache.

Replacement algorithms. When the cache space is full, the existing cached items must be moved from the cache for the newly inserted write requests. When the cache replacement occurs, LCache first replaces the clean data page from the tail of the clean data LRU list, which can minimize the cost of cache replacement. When there is no clean data page, it will continue to replace from the tail of the dirty LRU list.

4.3 Read and Write Operation

After user-level read and write requests arrive at the device, the HIL is first splitted into flash transactions. The index of the cache linked list will be queried, meanwhile, the corresponding operation will be executed with reference to hit or not.

If flash transactions belong to read requests, when cache hit occurs, if the hit data is dirty data, it will be moved to the head of the dirty LRU list. If it is clean data, it will be moved to the head of the clean LRU list. When cache

Table 1. Details of the simulation parameters.

SSD Organization	Host interface: PCIe 3.0 (NVMe 1.2) Total Capacity: 24 GB 6 channels 4 chips-per- channel
Flash Microarchitecture	4KiB page, 448 B metadata-per-page, 64 pages-per-block, 1024 blocks-per-plane, 2 planes-per-die, 2 die-per-chip
Flash Latencies[10]	Read latency: 75 us, Program latency: 750 us, Erase latency: 3.8 ms
Flash translation layer (FTL)	GC Policy: Greedy GC Threshold: 0.05 Address Mapping: DFTL TSU Policy: Sprinkler[11] Overprovisioning Ratio: 0.07

miss occurs, the cache that cannot service the corresponding requests reads the corresponding data from the underlying flash memory.

If flash transactions belong to write requests, when cache hit occurs, the hit data is extracted from the corresponding LRU linked list and updated to new data. After modifying the corresponding metadata, it is inserted into the head of the dirty LRU linked list. Otherwise, the cache misses, if the cache space is not full, the new data page will be directly inserted into the head of the dirty-data LRU linked list and return the completion information. If the cache space is full, the cache should be triggered to replace the new data page to free up the cache space and the new data page are inserted into the dirty data LRU linked list head. Please refer to Section 4.2 for cache replacement algorithm algorithm.

5 Experimental setup and Evaluation

5.1 Experimental setup

We use an open-source simulator, MQSim [6], to simulate, evaluate and compare our proposed LCache LRU write cache strategy, CFLRU, and MQSim. MQSim, released by the cmu-safari research group, can accurately simulate the latest NVMe protocol solid-state disk prototype and the main front-end and back-end components of multi-queue solid-state disk (MQ-SSD). The default cache size in the experiment is 32MB. The simulation parameters are shown in Table I.

We use real enterprise-scale workload to study the performance of the cache scheme. We select several typical block-level workloads from UMass trace and Microsoft Cambridge Research.

5.2 Performance Evaluation

Response Time Response time refers to the time from the time when I/O request enters the SQ queue to the time when the host receives the return information from SSD. It is commonly used to evaluate the performance of storage

devices. Fig. 4 shows the response time of LCache under different workloads compared with LRU, CFLRU and MQSim under four-type traces. In proj_0, LCache achieves an improvement of 59.8%, 60%, and 14.81%, compared with LRU, CFLRU, and MQSim respectively. It enhances about 84%, 83.3%, and 26% under prxy_0 configured with the three cache managements. In stg_0, the response time of LCache is 1.1% higher than that of MQSim, but the erase count is 47.2% lower than that of MQSim.

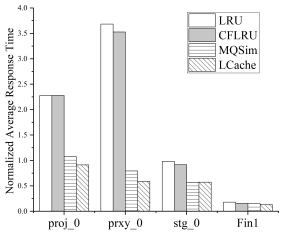

Fig. 4. Response Time of LCache, LRU, CFLRU, MQSim under traces.

Standard Deviation of response time The standard deviation of response time is used to evaluate the optimization of storage system performance cliff caused by different cache strategies, illustrated as Fig. 5. Compared with LRU, CFLRU, and MQSim under proj_0, the standard deviation of response time increases by 68.2%, 68%, and 30.2% respectively, and under prxy_0. Besides, LCache achieves 3%, 75. 1%, and 37. 2% improvement, respectively. LCache has a good performance in Fin1 and stg_0.

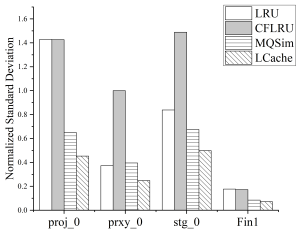

Fig. 5. Standard Deviation of Response Time of LCache, LRU, CFLRU, MQSim under traces.

Erase Count Erase count in flash memory intuitively reflects the SSD lifetime. Each block in flash memory has the upper limitation. Fig. 6 shows that LCache

has a slightly higher erase count compared with LRU and CFLRU. The reason is that LCache update mechanism can lead to much data written into flash memory compared with other strategies. However, LCache optimizes the erase count by 2.1%, 69.3%, 47.2%, and 45.4%, respectively, under proj_0, prxy_0, stg_0, and Fin1, compared with MQSim.

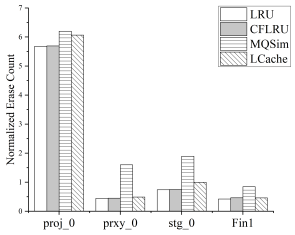

Fig. 6. Erase Count of LCache, LRU, CFLRU, MQSim under traces.

6 Conclusion

With the space increment of an SSD, the cache offered inside the SSD is limited. In the paper, we propose a machine learning-based cache management, LCache, for NDP-based SSD. LCache employs decision tree C4.5 to divide I/O requests into two types, (1) data that will re-access the cache; (2) data that will not re-access the cache. The cache data is dynamically updated into flash memory in pursuant with the load status of flash memory chip through the active update mechanism. LCache effectively balances the access of I/O requests to flash memory, thereby improving the performance of the whole SSD. We simulate the performance of LCache through real-world traces and the result shows that LCache outperforms other existing caching strategies.

References

1. Hua Wang, Xinbo Yi, Ping Huang, Bin Cheng, and Ke Zhou. Efficient ssd caching by avoiding unnecessary writes using machine learning. In *Proceedings of the 47th International Conference on Parallel Processing*, pages 1–10, 2018.
2. Rajeev Balasubramonian, Jichuan Chang, Troy Manning, Jaime H Moreno, Richard Murphy, Ravi Nair, and Steven Swanson. Near-data processing: Insights from a micro-46 workshop. *IEEE Micro*, 34(4):36–42, 2014.
3. Yang Seok Ki et al. In-storage compute: An ultimate solution for accelerating i/o-intensive applications. *Flash Memory Summit*, 2015.
4. Richard L. Mattson, Jan Gecsei, Donald R. Slutz, and Irving L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems journal*, 9(2):78–117, 1970.

5. Seon-yeong Park, Dawoon Jung, Jeong-uk Kang, Jin-soo Kim, and Joonwon Lee. Cflru: a replacement algorithm for flash memory. In *Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 234–241, 2006.
6. Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. Mqsim: A framework for enabling realistic studies of modern multi-queue ssd devices. In *16th USENIX Conference on File and Storage Technologies FAST 18*, pages 49–66, 2018.
7. Umass trace repository. http://traces.cs.umass.edu/index.php/Storage/Storage, 2020.
8. Snia traces. http://iotta.snia.org/traces/, 2020.
9. Jen-Wei Hsieh, Tei-Wei Kuo, and Li-Pin Chang. Efficient identification of hot data for flash memory storage systems. *ACM Transactions on Storage (TOS)*, 2(1):22–40, 2006.
10. Nand flash memory mlc mt29f256g08ckcab datasheet. Micron Technology, Inc, 2014.
11. Myoungsoo Jung and Mahmut T Kandemir. Sprinkler: Maximizing resource utilization in many-chip solid state disks. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 524–535. IEEE, 2014.