A Write-Efficient Cache Algorithm based on Macroscopic Trend for NVM-based Read Cache

Ning Bao, Yunpeng Chai
Key Laboratory of Data Engineering and Knowledge Engineering, MOE
School of Computing, Renmin University of China
Beijing, China
{baoning, ypchai}@ruc.edu.cn

Xiao Qin
Samuel Ginn College of Engineering
Auburn University
Auburn, USA
xqin@auburn.edu

***Abstract*—Compared with traditional storage technologies, non-volatile memory (NVM) techniques have excellent I/O performances, but high costs and limited write endurance (e.g., NAND and PCM) or high energy consumption of writing (e.g., STT-MRAM). As a result, the storage systems prefer to utilize NVM devices as read caches for performance boost. Unlike write caches, read caches have greater potential of write reduction because their writes are only triggered by cache updates. However, traditional cache algorithms like LRU and LFU have to update cached blocks frequently because it is difficult for them to predict data popularity in the long future. Although some new algorithms like SieveStore reduce cache write pressure, they still rely on those traditional cache schemes for data popularity prediction. Due to the bad long-term data popularity prediction effect, these new cache algorithms lead to a significant and unnecessary decrease of cache hit ratios. In this paper, we propose a new Macroscopic Trend (MT) cache replacement algorithm to reduce cache updates effectively and maintain high cache hit ratios. This algorithm discovers long-term hot data effectively by observing the macroscopic trend of data blocks. We have conducted extensive experiments driven by a series of real-world traces, and the results indicate that compared with LRU, the MT cache algorithm can achieve 15.28 times longer lifetime or less energy consumption of NVM caches with a similar hit ratio.**

I. Introduction

In recent years, non-volatile memory (NVM) techniques are evolving rapidly. Flash-based solid-state drives (SSDs) [1] have been widely deployed in many enterprise environments. In addition, PCM [2] and STT-MRAM [3] are on their ways. For example, Intel and Mircon have launched their 3D XPoint products [4] which are speculated to adopt PCM technology.

However, most of these emerging NVMs have their own problems in writing. For example, Flash and PCM have limited write endurance compared with DRAM. Each cell of Multi-Layer Cell (MLC) Flash chips can only be updated for 5,000 $\sim$ 10,000 times before wearing out [5]. Intel's new-generation Optane Memory (i.e., 3D XPoint) just achieves 3 times longer write endurance of NAND Flash [4]. Moreover, the writing cells of STT-MRAM consume too much energy, about 100 times higher than DRAM [6]. Therefore, as a new storage technique with excellent I/O performances, but high costs and write problems, NVM devices are more appropriate as a read cache in storage systems.

For such an NVM-based read cache, existing cache algorithms cannot reduce the write pressure without a noticeable degradation of cache hit rates. Traditional caching algorithms such as LRU and LFU update very frequently because they cannot predict data hotness in the long future. They lose a mass of history information and predict only according to a single value reflecting recency or frequency. Some studies have proposed cache algorithms to reduce write pressure on NVM devices, such as SieveStore [7] and LARC [8]. However, these approaches still rely on traditional cache algorithms (e.g. LRU, LFU, etc.) to predict data popularity, just keeping some data in cache for a longer time for write reduction. Predicting long-term popularity in this way is not accurate and the performance of these existing cache algorithms is not optimal.

In this paper, we proposed a new cache algorithm called Macroscopic Trend (MT), which employs a new macroscopic view to observe the trend of the data access history. It can discover long-term hot data more accurately based on the observed macroscopic trend, and thus keep these data in cache for a long time to reduce NVM writes. Therefore, MT can achieve relatively higher hit rates under low data updating frequency compared with existing solutions. We conducted extensive experiments driven by a series of real-world traces, and found that compared with LRU, the MT cache algorithm can achieve 15.28 times longer lifetime or smaller energy consumption and increase the hit rates by 24.03% on average.

II. Macroscopic Trend Cache Algorithm

A. Motivation

NVM-based read caches should employ cache algorithms which can identify long-term hot data accurately so that the NVM update frequency can be significantly reduced. However, LRU and LFU cannot effectively predict data popularity in the long future because they lose too much data access information. To acquire more information and avoid information

This work is supported by the National Key Research and Development Program of China (No. 2018YFB1004401), National Natural Science Foundation of China (No. 61732014, 61472427, and 61572353), Beijing Natural Science Foundation (No. 4172031), the National Science Foundation of Tianjin (17JCYBJC15400), the Fundamental Research Funds for the Central Universities and the Research Funds of Renmin University of China (No. 16XNLQ02), and open research program of State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Science (No. CARCH201702) . The corresponding author of this paper is Yunpeng Chai.

978-3-9819263-2-3/DATE19/©2019 EDAA

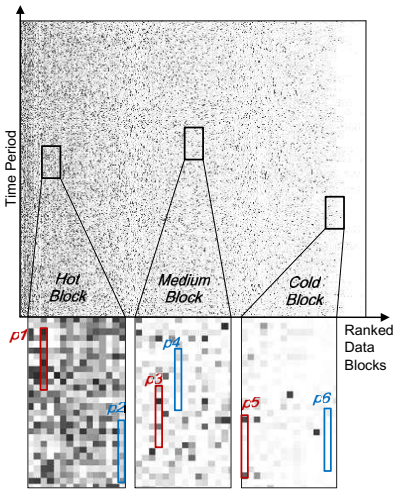

Fig. 1. Popularity distribution of sampled data blocks in 10,000-request period for *websearch*

explosion, we observe the access history of data blocks in a coarse granularity, i.e., recording the access counts in some adjacent time periods for each block, as shown in Fig. 1.

In this figure, the trace *websearch* is divided into a list of periods, and each period contains 10,000 requests. Each column corresponds to one data block, and the Y axis represents time periods. The darker points reflect more accesses in a period. All the blocks are ranked in a descending order according to the total access counts. Such a macroscopic view in Fig. 1 shows that the most-accessed blocks in the left side experience a lot of deep-color periods distributed in the service time.

However, the hot data also have some cold stages, such as marked periods in the frame *p2*. Moreover, the medium data and the cold data may have hot stages, such as *p3* and *p5*. In this case, it is hard for either LRU or LFU to distinguish hot, medium, and cold data with each other because it only relies on the latest access time or the total access count and discards all the details. Thus we proposed to observe data popularity in such a new macroscopic trend perspective.

B. Overview

Based on the macroscopic trend observation, we proposed a new cache algorithm called Macroscopic Trend (MT) to make accurate predictions on long-term hot data. Fig. 2 illustrates the architecture of MT, which is composed of three parts:

The *macroscopic observation* module records the access counts of the requested blocks in a coarse granularity to obtain the popularity distributions of accessed blocks. The small squares with different gray levels on the left-hand side of Fig. 2 represent the access count of each period. A dark color means a high access count.

The *multi-pattern evaluation* module matches the access-count distribution with some known long-term hot (LTH) data patterns (e.g., high proportion of active periods) to measure the data popularity in the future. According to the count of the matched LTH patterns, the blocks are placed into different positions of an NVM queue (NQ) for the already cached blocks in NVMs or a candidate queue (CQ), which buffers potential long-term hot data for cache replacement.

Finally, the *lazy updating* module changes the cached data updating mode from the request level to the period level, reducing the NVM writes significantly. There is a limit quota of updating cached blocks for each round.

C. Lightweight Macroscopic Observation

This module is designed to form a macroscopic popularity distribution by recording the total access counts of each period. The *Lightweight Macroscopic Observation* module contains a record of the access counts of multiple history periods and a current period for each accessed block as shown in Fig. 2. The data structure is implemented as a hash table in RAM and is updated per request. The period length of MT is usually set as tens of thousands of user requests.

D. Multi-Pattern Evaluation

The LTH patterns in MT were extracted based on our observation and experience. Take Fig. 1 for example. The most-accessed blocks (e.g. *p1*) usually have many active periods (i.e., dots in the dark colors). However, for the medium hot block (e.g., *p3*) and the cold block (e.g., *p5*), they have significantly fewer active periods in the time window, although they may have similar total access counts. Since the distribution of active periods is useful to distinguish hot data, three patterns measuring the macroscopic trend of active periods are adopted, namely Stability-Pattern (S-Pattern), Continuity-Pattern (C-Pattern) and Irregularity-Pattern (I-Pattern).

S-Pattern. S-pattern selects stably hot data blocks, whose percent of the active periods in a time window needs to be large enough.

C-Pattern. A block which has multiple continuous active periods matches C-Pattern, which is hard for the medium hot or cold data to match.

I-Pattern. A block is considered to match this pattern when the block's total access count in the past time window is larger enough, no matter how its periodical access counts distribute.

The thresholds of the three patterns are self-adaptive in MT. Take I-Pattern for example and suppose T blocks are updated each time. All the non-cached blocks are sorted based on their total access counts and the access count of the T_{th} block is the threshold of I-Pattern. After evaluating the counts of the LTH patterns that blocks satisfy, MT discards non-cached

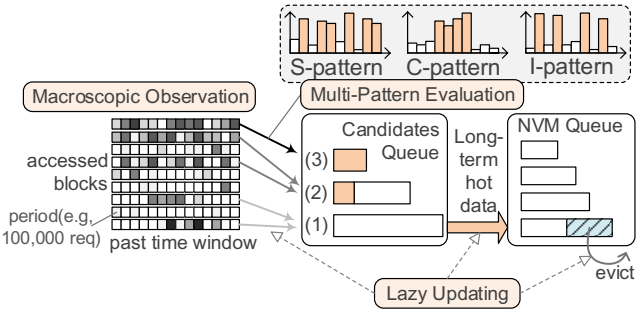

Fig. 2. The architecture of our proposed MT cache algorithm.

blocks that do not match any pattern; the other blocks are placed into the different subqueues of CQ or NQ according to their numbers of satisfied patterns. Of course, the more LTH patterns a block matches, the more likely the block is to be kept in or fetched into NVM caches.

E. Lazy Updating

In MT, we perform a lazy data updating manner, i.e., a certain amount of cached blocks will be replaced with new ones every one or multiple periods, because most cached data in NVMs can keep hot for a long time. The maximum number of the updated blocks each time is limited by a value of *lazy updating throttling*, denoted as T, which is used to avoid an excessive number of unnecessary data updates.

When a data updating process is triggered, the top T blocks of the CQ are fetched into NVM, resulting in evicting the worst data blocks in NQ with the same amount. Thus we need to get the best T blocks in CQ and the worst T blocks in NQ. In both CQ and NQ, blocks are first sorted according to the number that blocks satisfy LTH patterns, and then blocks in the same subqueue are sorted according to LFU.

F. Overhead

MT has a low space requirement because it only stores the metadata information in memory. For example, assuming each block is 16 KB and the system is smaller than 16PB, the extra space overhead for MT is the history table which only needs 13.2MB to store 660,000 requests. The NQ and CQ are temporary data structures and the space overhead of them are similar with the history table.

As for time overhead, the time complexity of recording the history informaiton is $O(1)$. Though the cache updates are costly, such operations happen infrequently and can be run in the background. So the time overhead is acceptable.

III. Evaluation

We conducted extensive experiments based on a trace-driven simulated caching system. The experiments were performed in a Dell R720 server coupled with Linux CentOS 7.4.1708, Intel Xeon CPU E5-2640 2.50GHz, 32GB DRAM, an enterprise-level 800GB Memblaze Q520 PCIe Solid State Drive (SSD) as the read cache, and a 2-TB Hard Disk Drives (HDDs) as the data storage.

Besides MT, we also implemented some representative cache algorithms in the simulation system including LRU, SieveStore, and period-LRU (pdLRU). LRU is a widely utilized cache algorithm and SieveStore is a good representative of the write optimized cache algorithms for NVM caches. As a common-sense cache algorithm, pdLRU employs LRU for data popularity prediction and updates the same amount of data as MT. Microsoft Research (MSR) Cambridge traces [9] collected from typical enterprise data centers were replayed in our experiments. Our evaluations cover all the 12 application categories [1] of the MSR-Cambridge traces.

[1]Including traces *usr, proj, prn, hm, rsrch, prxy, src, stg, ts, web, mds, wdev*

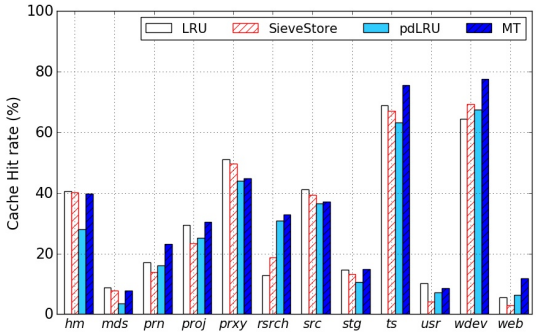

Fig. 3. Overall cache hit ratio results.

A. Overall Performance

First, we present the overall performance of MT along with other algorithms. Figs. 3 and 4 respectively illustrate the hit rates and the NVM write amounts of read caches. The cache size is set to a default value, i.e., 10% of the working set size. The default period length of MT is 100,000.

As Fig. 3 plots, in most cases, MT achieves either the highest (e.g., *prn*, *proj*, *rsrch*, *stg*, *ts*, *wdev*, and *web*) or very close to the highest (e.g., *hm*, *mds*, and *usr*) cache hit rates. The average hit ratio promotion of MT is 24.0% compared with LRU, 50.2% compared with SieveStore and 35.2% compared with pdLRU.

Fig. 4 exhibits that the NVM write amounts of MT are always much smaller than LRU and SieveStore for all the traces. Note that the NVM write amounts in Fig. 4 have been normalized based on MT's write amounts for each trace. MT reduces write amount by 15.28 times for LRU and 6.75 times for SieveStore on average.

In summary, the results plotted in Figs. 3 and 4 confirm that MT is conducive to discovering the long-term hot data, thereby reducing NVM write amounts significantly and achieving similar or even higher cache hit ratios at the same time.

B. Results under Different Cache Sizes

In this part, the experimental results under different NVM cache sizes driven by *wdev* are given. The ratios between cache sizes and working set sizes range from 5% to 25%. The hit ratios of all the cache algorithms increase along with the ascending cache sizes. Fig. 5 reveals that the hit rates of MT for trace *wdev* are higher than the others in all cases.

Fig. 6 exhibits the normalized NVM write amounts. The results indicate that MT significantly reduces the NVM write loads in all the cases except for pdLRU.

Along with the increase of cache size, the write amounts of the LRU and SieveStore keep declining. This trend is attributed by increasing hit rates. MT limits the allowed counts of block updating proportional to the cache size; therefore, MT's write amount increases as the cache size goes up. In fact, the scheme of MT is more appropriate than the others, because a large cache device can stand more write pressure than a small one.

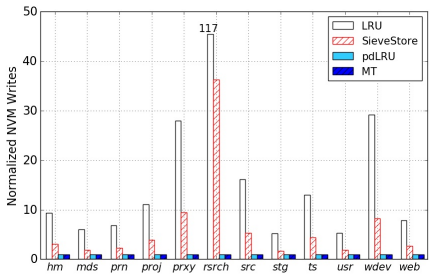

Fig. 4. Overall write amount results.

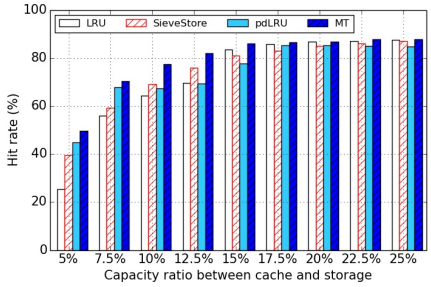

Fig. 5. Hit ratios under different cache sizes for *wdev*.

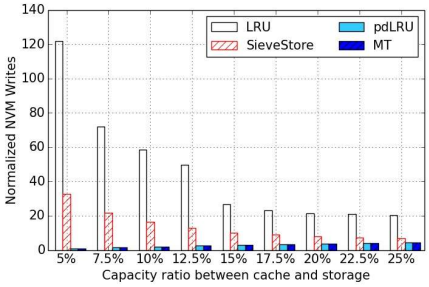

Fig. 6. Write amounts under different cache sizes for *wdev*.

IV. Related Work

Traditional caching algorithms are aimed to achieve high hit rates without considering data write amount, thus this kind of cache algorithms should not be directly used on the emerging NVM devices with write limitations. In recent years, some studies [7], [10] have focused on applying write optimized cache algorithmes on NVM-based caches. The existing solutions mainly include the following three categories:

Data Filtering Schemes. The first category schemes adopt a data filtering method to improve the quality of cached data and meanwhile to reduce the frequency of updating the cached data, such as SieveStore [7] and L2ARC [10]. SieveStore [7] keeps a record of the access counts of blocks and only caches those who have been accessed multiple times. However, these algorithms select data based on traditional cache schemes, leading to unpredictable system performance and declined hit ratios. In fact, our proposed MT algorithm falls into this data-filtering category, so we compare MT with the typical SieveStore algorithm in the evaluation part (Section III).

Delayed Eviction Schemes. The second kind of schemes reduce the NVM write amounts by extending the data caching time to avoid too early eviction of popular data, such as WEC [11] and ETD-Cache [12]. However, these algorithms also rely on traditional algorithms like LRU to select data.

Container-level Cache Schemes. The third group of cache schemes aim to lengthen the lifetime of SSD caches by reducing the inner write amplification rate of Flash chips, such as RIPQ [13] and SRC [14]. These solutions are only effective for Flash-based SSDs instead of other NVM devices.

V. Conclusions

For an NVM-based read cache with limited write endurance or huge write energy consumption, instead of traditional caching strategies that cause heavy write loads, a better solution is to fill it with long-term hot data to avoid the requirement of frequent cache updating. In this paper, we propose a new MT cache replacement algorithm, which keeps track of the macroscopic popularity trends and performs a lazy updating manner to reduce the write amounts of NVM-based read caches significantly and maintains a high hit rate. The experimental results driven by real-world traces reveal that MT is conducive to extending the NVM device lifetime or reducing the writing energy consumption by a factor of 15.28 on average while achieving higher hit ratios in more than half of the evaluation cases (24% higher on average) compared with LRU.

References

[1] A. Leventhal, "Flash storage memory," *Communications of ACM*, vol. 51, no. 7, pp. 47–51, 2008.

[2] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3. ACM, 2009, pp. 2–13.

[3] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto *et al.*, "A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-ram," in *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*. IEEE, 2005, pp. 459–462.

[4] P. Alcorn, "3d xpoint ssd pictured, performance and endurance revealed at fms," 2016, http://www.tomshardware.com/news/intel-micron-3d-xpoint-memory,32434.html.

[5] L. Grupp, J. Davis, and S. Swanson, "The bleak future of nand flash memory," in *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*, San Jos, CA, 2012.

[6] T. Hirofuchi and R. Takano, "Raminate: Hypervisor-based virtualization for hybrid main memory systems," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*. ACM, 2016, pp. 112–125.

[7] T. Pritchett and M. Thottethodi, "Sievestore: a highly-selective, ensemble-level disk cache for cost-performance," in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 163–174.

[8] S. Huang, Q. Wei, J. Chen, C. Chen, and D. Feng, "Improving flash-based disk cache with lazy adaptive replacement," in *Proceedings of the 29th International Conference on Massive Storage systems and Technology (MSST'13))*, 2013.

[9] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage (TOS)*, vol. 4, no. 3, p. 10, 2008.

[10] B. Gregg, "L2arc," 2008, https://blogs.oracle.com/brendan/entry/test.

[11] Y. Chai, Z. Du, X. Qin, and D. A. Bader, "Wec: Improving durability of ssd cache drives by caching write-efficient data," *IEEE Transactions on Computers*, vol. 64, no. 11, pp. 3304–3316, 2015.

[12] N. Dai, Y. Chai, Y. Liang, and C. Wang, "Etd-cache: an expiration-time driven cache scheme to make ssd-based read cache endurable and cost-efficient," in *Proceedings of the 12th ACM International Conference on Computing Frontiers*. ACM, 2015, p. 26.

[13] L. Tang, Q. Huang, W. Lloyd, S. Kumar, and K. Li, "Ripq: Advanced photo caching on flash for facebook." in *FAST*, 2015, pp. 373–386.

[14] Y. Oh, E. Lee, C. Hyun, J. Choi, D. Lee, and S. H. Noh, "Enabling cost-effective flash based caching with an array of commodity ssds," in *Proceedings of the 16th Annual Middleware Conference*. ACM, 2015, pp. 63–74.