2022 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES) | 978-1-6654-7296-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/CASES55004.2022.00010

Work-in-Progress: ExpCache: Online-Learning based Cache Replacement Policy for Non-Volatile Memory

Jinfeng Yang*, Bingzhe Li†, Jianjun Yuan§, Zhaoyan Shen¶, David Du*, and David Lilja*
*University of Minnesota, Twin Cities; †Oklahoma State University; §Expedia Group; ¶Shandong University
yang3116, yuanx270, du, lilja@umn.edu; bingzhe.li@okstate.edu; shenzhaoyan@sdu.edu.cn

***Abstract*—As emerging memory technologies (e.g., non-volatile memory (NVM)) coming out and machine learning algorithms successfully applying to different fields, the potentials of cache replacement policy for NVM-based systems with the integration of machine learning algorithms are worthy of being exploited to improve the performance of computer systems. In this work, we proposed a machine learning based cache replacement algorithm, named ExpCache, to improve the system performance with NVM as the main memory. By considering the non-volatility characteristic of the NVM devices, we split the whole NVM into two caches, including a read cache and a write cache, for retaining different types of requests. The pages in each cache are managed by both LRU and LFU policies for balancing the recency and frequency of workloads. The online *Expert* machine learning algorithm is responsible for selecting a proper policy to evict a page from one of the caches based on the access patterns of workloads. In experimental results, the proposed ExpCache outperforms previous studies in terms of hit ratio and the number of dirty pages written back to storage.**

***Index Terms*—Online learning, Cache policy, Non-volatile memory**

I. Introduction

With the rapidly increased volume of newly generated data, the demand for high-performance memory media becomes more critical than before. People pursue a high access bandwidth and expect a large density and non-volatility property on memory devices. As one of the emerging memory technologies, non-volatile memory (NVM) provides both the competitive performance as DRAM meanwhile and the property of non-volatility as storage devices [1]. Because all data reside in NVM is persistent, there is no need to write updated pages (i.e., dirty pages) back to storage for preventing data loss while power failure, which significantly increases the system performance. According to those advantages, NVM is expected to replace DRAM to serve as the main memory in the near future. The cache replacement policy determines which data should be evicted from NVM to storage devices so that the whole system can benefit the data reuse in NVM.

In order to fully take the performance advantages of NVM, designing a proper cache algorithm to manage the data elements within NVM becomes important. The cache algorithms have been intensively investigated over time. However, those algorithms were developed to make a replacement decision based on certain pre-defined rules (e.g., LRU algorithm always evicts the least recently used page when a replacement is required) instead of the real access patterns of a workload. Since these pre-defined rules are not adapted to the dynamically changed workload access patterns, these cache algorithms suffer significant performance degradation. This explains why conventional cache algorithms [2], [3], [4] only work well on a specific set of workloads.

In this paper, we propose a new online learning-based cache replacement algorithm, called ExpCache, for NVM-based systems to overcome the shortcomings of prior studies. The goals of ExpCache are improving both read and write hit ratios of NVM meanwhile eliminate the number of dirty pages written back to the storage. In the system, the real cache (i.e., NVM main memory) is divided into a read cache and a write cache for retaining different types of requests. Two experts, including an LRU policy and an LFU policy, are employed to manage the data elements within each of the real caches to balance both recency and frequency factors. In addition, two ghost caches are built to store the evicted pages from the real caches and emulate the LRU and LFU policies independently. The time-dependent *online Expert* algorithm continuously monitors the hit ratio inside those ghost caches and uses them as an indicator of workload access patterns. Finally, using the selected eviction policy, ExpCache sets evicting a page from the read cache having a higher priority while a certain condition is met to reduce the number of dirty pages written back to the storage.

II. ExpCache Algorithm

In this section, we provide a detailed description of the ExpCache algorithm. We introduce the design principle and the internal architecture of ExpCache, and its working processes at first. Two key components of ExpCache, including the time-dependent online Expert and read-write discriminated eviction policy, are discussed then.

Two fundamental factors (i.e., recency and frequency) are widely used in cache policies. This is because the patterns of most workloads follow either recency or frequency manner in one period of time. Thus, ExpCache is proposed by assuming that at every time step, there has the best strategy among two fundamental experts, recency expert (i.e., LRU policy) and frequency expert (i.e., LFU policy). The goal of the ExpCache

2643-1726/22/$31.00 ©2022 IEEE
DOI 10.1109/CASES55004.2022.00010

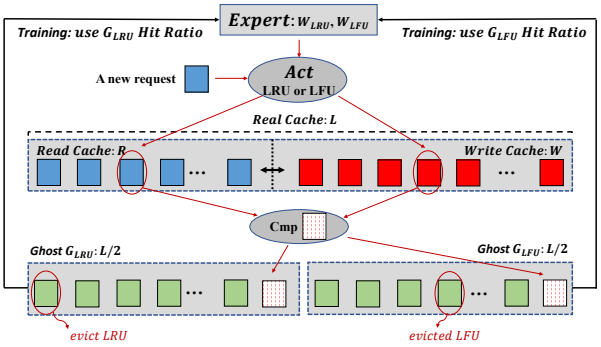

Fig. 1: ExpCache Architecture.

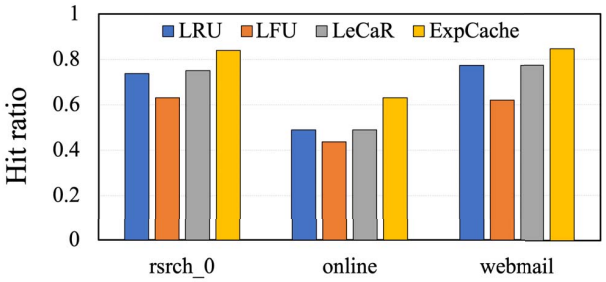

Fig. 2: Hit Ratio comparison between ExpCache and other baselines.

is to learn the caching environment, such as dynamically changed workloads and hit ratios, and then select a proper eviction scheme from the experts for the system. In order to map the *online expert algorithm* into the proposed policy properly, meanwhile consider the non-volatile property of NVM, the architecture of the ExpCache is carefully designed.

As shown in Figure 1, in the system, we split the real cache into two small caches, including a read cache denoted as *R* and a write cache denoted as *W*, for retaining different types of requests. Pages within the read cache are clean (i.e., have never been modified). Once a page is modified (i.e., the page becomes dirty), it will be moved into the write cache. In order to perform a replacement efficiently using any one of the two experts, the pages within each small cache are managed by both LRU and LFU-based structures.

Two ghost caches denoted as G_{LRU} and G_{LFU}, are built to simulate the behaviors of LRU policy and LFU policy independently. Each ghost cache records the metadata (i.e., page identifier) of the most recent evictions from the real cache. When a victim is evicted from the real cache, it is inserted into both G_{LRU} and G_{LFU}. The elements inside the G_{LRU} are strictly organized in an LRU order based on their timestamps inherited from the real cache. This can be done by using a heap data structure. Here, the timestamp of a page is the access number of the last access to that page. The time complexity of adding or deleting an element into or from the G_{LRU} is $log_2 N$. Where, N is the number elements within G_{LRU}. Since N is a constant and is usually much smaller than the number of requests within a workload, the overhead on operating G_{LRU} is negligible. Similarly, the elements inside G_{LFU} are strictly organized in an LFU order based on their corresponding access frequency and timestamp inherited from the real cache.

III. System Evaluation

In this section, we evaluate the effectiveness of our purposed design. Four baseline caching polices, including LRU, LFU, LeCaR [5], and H-ARC [6], are used for the comparison. Two types of workloads, including MSR [7] and FIU [8], are used for testing those cache polices with 20% cache size. As shown in Figure 2, the proposed ExpCache achiever higher hit ratio compared to prior work. This is because that ExpCache follows the workload patterns to cleverly evict proper pages to obtain higher hit ratios.

IV. Conclusion

In this work, we propose an ExpCache algorithm for computer systems that use NVM as the main memory. By considering recency and frequency factors and the non-volatility characteristics of the NVM device, we split the whole cache into two small caches (i.e., a write cache and a read cache) for preserving different types of requests. Each cache is managed by both LRU and LFU policies for balancing recency and frequency factors. The experimental results show that ExpCache delivers significant performance improvement on cache hit ratio compared with other baselines.

V. Acknowledgement

This work was partially supported by NSF I/UCRC Center Research in Intelligent Storage and the following NSF awards 1439622, 1812537, 2204656, and 2204657. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] J. Yang, B. Li, and D. J. Lilja, "Exploring performance characteristics of the optane 3d xpoint storage technology," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 5, no. 1, pp. 1–28, 2020.

[2] N. Megiddo and D. S. Modha, "Outperforming lru with an adaptive replacement cache algorithm," *Computer*, vol. 37, no. 4, pp. 58–65, 2004.

[3] S. Jiang and X. Zhang, "Lirs: An efficient low inter-reference recency set replacement policy to improve buffer cache performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 31–42, 2002.

[4] J. Yang, B. Li, and D. J. Lilja, "Heuristicdb: a hybrid storage database system using a non-volatile memory block device," in *Proceedings of the 14th ACM International Conference on Systems and Storage*, 2021, pp. 1–12.

[5] G. Vietri, L. V. Rodriguez, W. A. Martinez, S. Lyons, J. Liu, R. Rangaswami, M. Zhao, and G. Narasimhan, "Driving cache replacement with ml-based lecar," in *10th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 18)*, 2018.

[6] Z. Fan, D. H. Du, and D. Voigt, "H-arc: A non-volatile memory based cache policy for solid state drives," in *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2014, pp. 1–11.

[7] "Snia msr trace," http://iotta.snia.org/traces/388.

[8] "Snia fiu trace," http://iotta.snia.org/traces/390.