CPF: A Cross-Layer Prefetching Framework for High-Density Flash-based Storage

Longfei Luo[1,2], Han Wang[1,2], Dingcui Yu[1,2], Yina Lv[3], and Liang Shi[1,2],
longfei_luo@stu.ecnu.edu.cn, hanwang@stu.ecnu.edu.cn, dcyu@stu.ecnu.edu.cn, elainelv95@gmail.com, lshi@cs.ecnu.edu.cn
[1]Software/Hardware Co-design Engineering Research Center, Ministry of Education
[2]School of Computer Science and Technology, East China Normal University, Shanghai, China
[3]Department of Computer Science, City University of Hong Kong, Hong Kong, China

***Abstract*—The pseudo-single-level-cell (pSLC) technique is widely adopted in high-density flash-based storage to mitigate the performance and endurance problem of high-density flash memory. Furthermore, prefetching schemes can compensate for performance differences among storage tiers. Existing prefetchers are implemented in the operating system (OS) or storage layers. However, OS layer prefetchers are conservative since it is a challenge to achieve both high accuracy and large coverage simultaneously. Storage layer prefetchers are sub-optimal due to the performance differences between pSLC and DRAM. In this paper, a cross-layer prefetching framework (CPF) is proposed to prefetch data selectively. The basic idea is that high-accuracy data will be prefetched to DRAM and large-coverage data will be prefetched to the pSLC flash in storage. To make it practical, an adaptive regulator is further designed to dynamically adjust the cross-layer prefetching to ensure accuracy and coverage. Evaluations show that CPF can improve read performance and reduce data transfer costs significantly.**

I. Introduction

NAND flash memory has been developed from the single-level-cell (SLC) flash that stores one bit per cell to the quad-level-cell (QLC) with four bits per cell or even more for cost-effective considerations [1]. However, the performance of such high-density flash memory is degrading with its density improvement. For simplicity, QLC is selected as the representative of the high-density flash memory in this paper. To accelerate data accesses, the popular pSLC technique [2] is adopted, which is designed to transform part of QLC to SLC [3]. As a result, the performance of state-of-the-art high-density flash-based storage can be improved to that of SLC. To further mitigate the performance gap between device storage and host memory (i.e., DRAM), the prefetching technique is widely used to read data to DRAM ahead. Then, during accessing data, they can be read directly from the DRAM.

The design of prefetching has several concerns, including accuracy and coverage [4]. Previously, several prefetching schemes have been proposed for flash-based storage in different layers. On the OS side, Wu et al. [5] and Li et al. [6] proposed to optimize traditional readahead prefetching schemes which are sequential prefetch. Laga et al. [7] and Wang et al. [8] use the Markov model to predict the next data address. On the storage side, the Markov model is also used by Xu et al. [9] to prefetch data. Xu et al. [10] and Li et al. [11] predict subsequent accessing data by recording frequent access patterns. Ganfure et al. [12] and Chakraborttii et al. [13] utilize long short-term memory (LSTM) to prefetch data. However, the design of prefetchers is non-trivial due to cache pollution issues and input/output (I/O) bandwidth wastes caused by prefetching operations. Our evaluations reveal that it is hard to optimize the accuracy and coverage of prefetchers simultaneously. Therefore, existing prefetchers are generally conservative to avoid reading unuseful data, which limits the prefetching efficiency [4]. By analyzing the characteristics of OS-layer and storage-layer prefetchers, we find that coordination of both is an opportunity to optimize existing prefetchers.

In this work, a cross-layer prefetching framework, CPF, is proposed, which can make prefetchers achieve both high prefetching benefits and low costs with the help of the cooperation of DRAM in host and SLC flash in the high-density flash-based storage. The basic idea is to prefetch the data with high accuracy to DRAM and the data with low accuracy but high coverage to SLC flash. At this point, DRAM with high performance is used to store data with high accuracy without pollution, while SLC flash with large space is used to extend the prefetch coverage to gain more prefetching benefits. Since prefetching schemes generally have fluctuating accuracy and coverage, an adaptive regulator is further proposed to dynamically adjust the prefetcher. It is designed to optimize both the accuracy and coverage of DRAM and SLC flash. To the best of our knowledge, this is the first work proposing a cross-layer prefetching framework for pSLC technique-enabled high-density flash-based storage. Evaluations show that the proposed CPF helps the given prefetcher to achieve both high prefetching benefits and low costs. Data transfer costs between host memory and storage are reduced significantly while optimizing performance.

II. Background and Related Work

A. High-density flash-based storage

As the density of flash memory increases, the capacity is enlarged and the cost is reduced. However, its performance is reduced with the degraded reliability [14][15][16]. To address this problem, the pSLC technique [2] is widely adopted to provide high performance. Figure 1 shows an organization of

This work is supported by the NSFC (62141213, 62072177), Shanghai Science and Technology Project (22QA1403300), and East China Normal University Postgraduate International Conference Special Fund. The corresponding author is Liang Shi (lshi@cs.ecnu.edu.cn).

979-8-3503-4859-0/DATE24/© 2024 EDAA

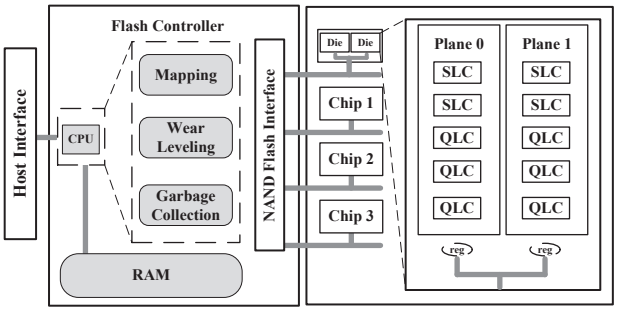


Fig. 1. Organization of high-density flash-based storage

the state-of-the-art high-density flash-based storage that adopts the pSLC technique. The flash controller is equipped with computation units to manage flash storage [17][18], including mapping, garbage collection (GC), wear leveling, etc. The flash chip array of the state-of-the-art high-density flash-based storage is organized as traditional flash storage except for the block organized in each plane. pSLC technique allows the flash controller to transfer part of QLC blocks to SLC blocks by storing only one bit per cell [2]. This leads to SLC blocks and QLC blocks per plane. For write requests, the flash controller will schedule them to SLC blocks to leverage their high performance and endurance [1]. When the device is idle or there are few free SLC blocks, data will be migrated to QLC blocks. To minimize write amplification, greedy GC [19] is widely used. Specifically, an SLC block with the most invalid data will be selected for recycling at first. Then the valid data in it will be migrated to QLC blocks. Finally, this SLC block will be erased to recycle space. Additionally, SLC flash is also used to accelerate data access by promoting data from QLC to it [20].

B. Prefetching on flash-based storage

Since there exists a significant performance gap between flash storage and DRAM, prefetching is a widely studied technique to improve user experience. By fetching data in advance, the latency of accessing these data will be reduced to the access latency of DRAM.

Existing prefetchers are implemented in two layers. On the OS side, readahead as a typical prefetching technique [5][6] has been integrated into the Linux kernel. It aims to optimize read performance when the accessing logical block addresses (LBA) are sequential. Apart from this, multiple studies proposed to prefetch data by prediction using historical information or machine learning methods. For example, Wang et al. [8] and Laga et al. [7] use the Markov model to forecast future read requests, to guide reading the relevant data in advance. On the storage side, Xu et al. [9] also use the Markov model to prefetch data. Ganfure et al. [12] use long short-term memory (LSTM) to predict the subsequent LBAs. Xu et al. [10] and Li et al. [11] try to mine frequent access patterns based on historical information.

There are two metrics to evaluate prefetching: accuracy and coverage. Accuracy is the fraction of useful prefetches among all issued prefetches, while coverage represents the proportion of useful prefetches in all demand reads. To improve accuracy, the prefetching scheme is generally conservative which only prefetch data that is most likely to be accessed next. Differently, coverage is improved by aggressive prefetching schemes which will prefetch a large amount of data. Obviously, there is a tradeoff between accuracy and coverage.

III. Problem Statement and Motivation

In state-of-the-art prefetching schemes, it is a dilemma to guarantee the prefetching accuracy and coverage at the same time, especially in consumer devices with limited computation and storage resources. In order to quantify the relationship between prefetching benefits and costs, we conducted a preliminary study by using the Markov model as a representative prefetcher. This is because the Markov model is implemented in both the OS layer [7][8] and the storage layer [9], which helps us to fully understand the benefits of prefetching.

A. Markov model-based prefetcher

Markov model is a lightweight probabilistic machine that learns and predicts I/O access patterns with minimum computation and storage consumption. A Markov model is a sequence S of random variables $S = LBA_1, ..., LBA_N$ with the Markov property, namely that the probability of moving to the next LBA depends only on the present LBA.

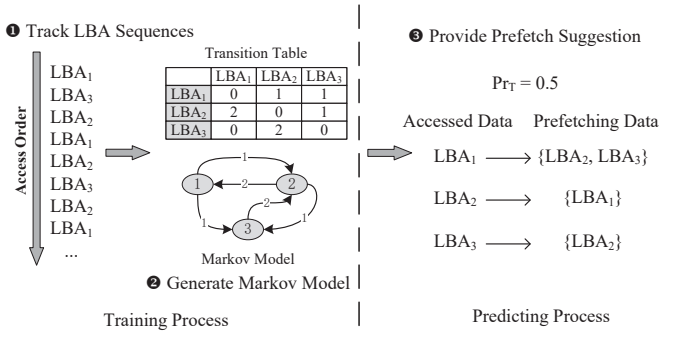


Fig. 2. An example of the Markov model-based prefetcher.

Figure 2 shows an example of the Markov model-based prefetcher. During the training process, a transition table is constructed where rows are labeled with the current LBA and columns are labeled with their next LBA. The number of occurrences $O_{i,j}$ is held for each transition from a page LBA_i to the next page LBA_j. By dividing the number of occurrences by the total number of transitions, we can get the probability of transition $Pr_{i,j}$. For example, the total number of transitions of LBA_1 is 2, and the number of occurrences $O_{1,2}$ and $O_{1,3}$ are both 1. Therefore, $Pr_{1,2}$ and $Pr_{1,3}$ are both 50%. During the prediction process, based on the current accessed data, the probability of the transition is not smaller than a threshold P_T will be chosen. For instance, if Pr_T is 50%, LBA_2 and LBA_3 will be prefetched when LBA_1 is accessed.

B. Benefits and costs of prefetching

Since Pr_T can influence the aggressiveness of the prefetcher, we adjust it to show its impacts on accuracy and coverage. We evaluate real-device workloads from the industry and run in an in-house simulator that integrates a Markov-based prefetcher and storage of consumer devices. The detailed experimental setup is in Section V. For each workload, the first half is used to train the prefetcher and the second half is used to evaluate the prefetcher. Figure 3 shows the collected accuracy and coverage

of workload *W-1* by varying the probability of the transition. The conclusion for other workloads is similar.

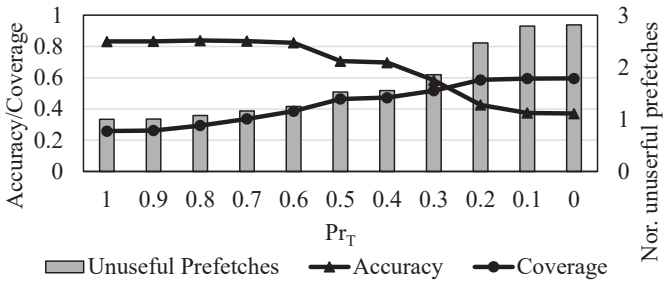


Fig. 3. Trade-off between accuracy and coverage.

First, with Pr_T decreases from 1 to 0, the accuracy is decreased from 83% to 37%, but the coverage is increased from 25% to 60%. This is because a higher Pr_T only allows the data that has a high possibility to be accessed in the near future to be prefetched, while lower Pr_T will allow more data to be prefetched. It reveals a ***Problem:*** *The accuracy and coverage of the prefetcher are hard to be high simultaneously.*

Second, with increasing the coverage, the number of unuseful prefetches increases by 2.81x. If prefetching these data to DRAM, the main problem is much unuseful prefetched data may pollute DRAM. Specifically, some useful data stored in DRAM are required to be evicted to store these unuseful prefetched data due to DRAM being generally small. SLC is also widely used to speed up data access [1]. The large capacity of SLC makes it able to store some unuseful data without evicting other useful data. However, the performance gap between SLC and DRAM makes it sub-optimal to prefetch data with high accuracy to SLC. These motivate us ***Motivation:*** *Existing prefetchers can be optimized by coordinating OS layer and storage layer prefetchers.*

IV. Cross-layer Prefetching Framework

A. Overview

Inspired by two key problems of existing prefetching methods, this work proposes a cross-layer prefetching framework to selectively prefetching between DRAM and the SLC flash, ensuring high prefetching accuracy and coverage, as well as avoiding DRAM pollution. The basic idea is to ❶ prefetch the data with high accuracy to DRAM while ❷ the data with low accuracy to the SLC flash. However, there are several challenges that need to be resolved. First, how to distinguish prefetching data of different accuracy to determine the prefetching destination. Second, how to adapt CPF to changing user access patterns.

To solve these challenges, two models are integrated into the cross-layer prefetching framework. Its architecture is shown in Figure 4. A prefetcher is often implemented in the host system to accelerate data access by prefetching some useful data to DRAM. To solve the first challenge, a selector is placed between the prefetcher and the high-density flash-based storage. The selector can perform different prefetchings based on the suggestion of the prefetcher with different configurations which can affect the aggressiveness of the prefetcher. To address the second challenge, an adaptive regulator is further proposed. The regulator can adapt the accuracy and coverage of the

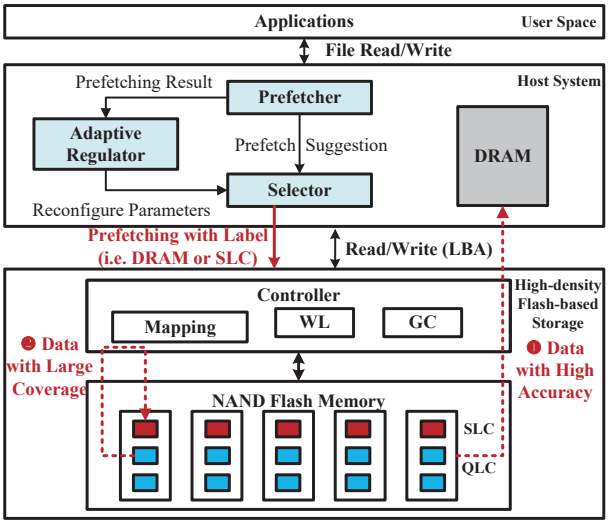


Fig. 4. Architecture of cross-layer prefetching selector

prefetching data in DRAM and SLC flash by adjusting the configurations for the selector based on the prefetching results of the current prefetcher.

B. Cross-layer Prefetching Selector

To better understand the design and benefits of the cross-layer prefetching selector, we take the Markov model as an example which has been introduced in Section III. Note that other prefetchers can also be easily adopted, like frequent access pattern-based prefetchers [10][11], LSTM [12], and so on. In Markov model-based prefetchers, Pr_T is the selective parameter to achieve selective prefetching since it can influence the aggressiveness of prefetchers.

Figure 5 presents the design of the cross-layer prefetching selector based on the Markov model. In the design, two Pr_T values (Pr_{high} and Pr_{low}) are set to decide whether to place the prefetched data in DRAM or SLC flash. On the one hand, if the transition probability is higher than Pr_{high} (❶), the corresponding pages will be prefetched to DRAM. By doing so, the precious capacity of DRAM can be fully used. On the other hand, if the transition probability is between Pr_{high} and Pr_{low} (❷), the corresponding pages will be prefetched to the SLC flash. Since the capacity of the SLC flash is much greater than DRAM, accuracy can be sacrificed to improve coverage. This leads to the number of prefetch hits will be increased and the benefit of prefetching is improved.

One challenge is that SLC flash is usually used as a write cache to accelerate write processes, and if too much data with low accuracy are prefetched to the SLC flash, the write performance will be affected. To solve this issue, Pr_{low} is used to limit the number of the data prefetched to the SLC flash and guarantee these data are valuable which may be used in the latter. At the same time, since SLC flash has high performance in writing data and the queue depth in the device is generally low [21], the conflict between prefetching data and normal writes is minor. Based on our experience, Pr_{high} and Pr_{low} are initially set to 80% and 20%, respectively with the best performance under the evaluated workloads.

The implementation of the cross-layer prefetching selector is presented on the right side of Figure 5. When the system call

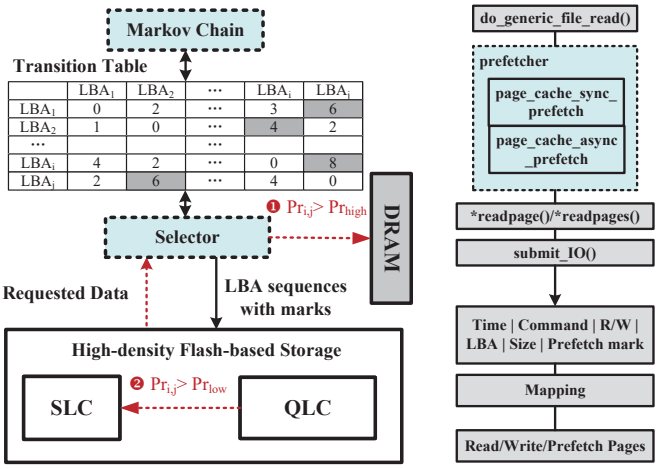


Fig. 5. The selector based on the Markov model

of file read is used, the prefetcher is activated. The prefetcher-specific system call is running to prefetch data. An integer marker (prefetch label) in *struct bio* is added to classify all host reads into three categories. Specifically, in this paper, 0 is for normal reads, 1 is for prefetching reads to DRAM, and 2 is for prefetching reads to the SLC flash. The cross-layer prefetching selector will attach this tag to the corresponding requests. Before issuing the prefetching requests, DRAM has reserved some space to store the prefetched data. When the storage receives prefetching requests, the corresponding mapping table is loaded to locate the data. For prefetching requests with tag 1, the corresponding data will be read and sent to the host. For prefetching requests with tag 2, if the data is in the QLC flash, it will be migrated to the SLC flash. Otherwise, nothing needs to be executed since the data has been in the SLC flash.

Adopting Selector for a Given Prefetcher: The proposed cross-layer prefetching selector is transparent to the prefetcher implemented in the host. The only modification that should be made is the selection of the configuration which controls the aggressiveness of the prefetcher. This kind of configuration can be found in most prefetchers [5][12][13][11][10], such as the adjustment of the prefetch window in readahead.

C. Adaptive Regulator

One important issue for the above selector is the configurations of Pr_{high} and Pr_{low}. Although they can guide the prefetching accuracy, the prefetching accuracy still may be lower than expected due to the performance fluctuation of the training model. Obsolete Pr_{high} and Pr_{low} may yield little benefits. On the one hand, if the accuracy is much lower than expected, DRAM will be polluted. Too much unuseful data are stored in SLC flash, which will influence the storage performance. On the other hand, if the accuracy is much higher than expected, the coverage has the potential to be increased further which can improve prefetching benefits. To optimize the cross-layer prefetching selector, an adaptive regulator is proposed to adjust the configurations.

First, Acc_{high} and Acc_{low} are used to represent the expected high and low accuracy, which are determined by two sets of information: 1) accuracy and the coverage of the prefetcher during the training process and 2) the performance and capacity of storage. The former can indicate what accuracy the model can achieve and how much data will be prefetched. The latter can evaluate the cost and benefit of the prefetching. For example, if the performance gap between SLC and QLC is large, the Acc_{low} should be set smaller since the benefit of prefetching hits is large but the cost of prefetching missing in DRAM is small. In this paper, Acc_{high} and Acc_{low} are set as 80 percent and 40 percent based on Figure 3, respectively.

Second, the accuracy and coverage of the prefetcher should be recorded where accuracy equals $\frac{Prefetching\ hits}{Prefethes}$ and coverage equals $\frac{Prefetching\ hits}{Host\ reads}$. The number of prefetches, normal reads, and prefetch hits of DRAM are easy to acquire and record during the I/O requests serving process. However, the number of prefetch hits of the SLC flash is transparent to the host. To solve this challenge, one of the reserved bits of the mapping table entry [22] is used to indicate that the page is a prefetching page. When serving read requests, a tag is sent back along with the requested data based on this bit. If the bit is set, the prefetcher will increase the number of prefetch hits of the SLC flash. When the prefetching pages are evicted to the QLC flash, the corresponding bit will be reset. The accuracy and coverage of DRAM and the SLC flash are held respectively to adjust Pr_{high} and Pr_{low}.

Finally, the regulation process is introduced in Algorithm 1. 1) The accuracy of DRAM and SLC flash is recorded within each time window and will be compared with Acc_{high} and Acc_{low}, respectively. 2) If the difference between them is higher or lower than a threshold D_T, Pr_{high} or Pr_{low} will be increased or decreased by a step size. In this paper, D_T is set to 10% and the step size is 0.1 based on Figure 3 for simplicity. 3) After the adjustment, if the accuracy is stable but the coverage is reduced significantly, the parameter will be recovered.

Algorithm 1: Regulation Process

```
if current_time − old_time > time_slot then
    if accuracy of DRAM, > Acc_high + D_T then
        old_Pr_high = Pr_high;
        Pr_high− = step_size;
    else if accuracy of DRAM < Acc_high − D_T then
        old_Pr_high = Pr_high;
        Pr_high+ = step_size;
    else if current_coverage < old_coverage − D_T then
        Pr_high = old_Pr_high;
    if accuracy of SLC > Acc_low + D_T then
        old_Pr_low = Pr_low;
        Pr_low− = step_size;
    else if accuracy of SLC < Acc_low − D_T then
        old_Pr_low = Pr_low;
        Pr_low+ = step_size;
    else if current_coverage < old_coverage − D_T then
        Pr_low = old_Pr_low;
```

The influence of CPF on GC: One additional concern for the above design is the influence of CPF on the GC of high-density flash storage. When the space of SLC flash is insufficient, greedy GC schemes [19] are adopted to recycle space. The block with the most invalid data will be selected and the valid data in it will be evicted to QLC flash. However, when CPF is adopted, the valid data in it may be data that was just prefetched. In this case, the effect of CPF will be reduced and the write amplification may be increased. To avoid

this problem, the expiration time of the prefetched data is adopted. First, the prefetched data and normal data in SLC are stored in the different blocks. Second, when the storage time of all prefetched data stored in an SLC block exceeds the expiration time, it is allowed to recycle this block. This is because prefetching data is mainly used to serve subsequent read requests. The time for storing prefetched data can be obtained by subtracting the write time recorded in the out-of-band area of the corresponding physical page from the current time. In this paper, the expiration time is 1 hour based on the characteristics of workloads.

D. Implementation and Analysis

On the OS side, the selector and regulator are added. Both of them are implemented based on the prefetcher. As introduced in Section IV-B, the selector is required to attach an integer marker, i.e. prefetch label, on the function path of the prefetcher. This is for distinguishing the type of prefetching requests. The selector will decide the prefetch label of each request based on the comparison between $Pr_{i,j}$, Pr_{high}, and Pr_{low}. These modifications are minor. The regulator maintains five parameters, including the number of prefetching hits in DRAM and SLC flash, total prefetches in DRAM and SLC flash, and the host reads. The space overhead of each of them is $\log_2 N$ where N is the value of the parameters. Therefore, this overhead is negligible.

On the storage side, to support the selector, the storage controller is required to recognize the prefetch label of each I/O request and prefetch data from QLC to SLC. The recognition of the prefetch label is easy to achieve by judging the value of the specified bit. The process of transferring data from QLC to SLC has been integrated into the storage controller generally which does not need to be modified. To support the regulator, when the prefetching hits in SLC flash, a signal along with the requests will be sent back to the host system. Since this signal is only a bit, the transfer cost is negligible.

Based on the above discussion, although the proposed CPF requires modifications to the OS and storage, they are minor and easy to implement. At the same time, the co-designs between the OS and storage are popular in industry and academia. For example, multi-stream methods [23] categorize data based on the update frequency and then send the category to the storage. The storage controller will put the data with the same category into the same block to improve GC efficiency. Kim et al. [24] try to attach the information to distinguish requests belonging to foreground or background apps for improving user experience. Luo et al. [22] transmit a tag in each request to identify critical data to improve their reliability. More and more co-designs disclose that the cooperation between the host system and storage can better improve user experiences than single-layer designs.

V. Evaluation

A. Experimental Setup

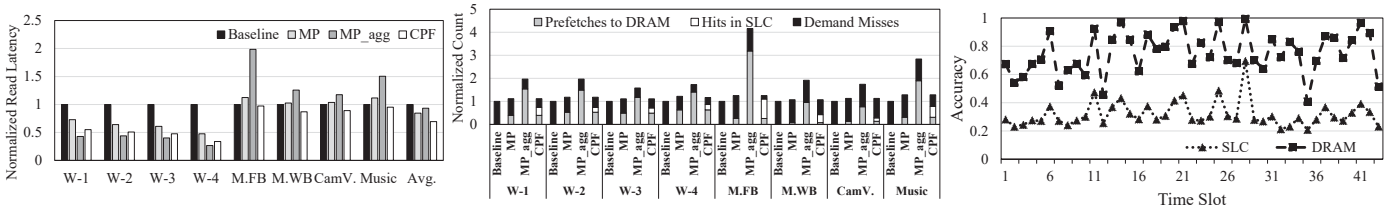
Due to the limitation of revisiting the storage controller, CPF is evaluated as follows: First, four industry-provided workloads collected from real mobile devices for several hours by blktrace [25] are used, named W-1 to W-4. Additionally, four read-dominate workloads from Nexus 5 [21] are also used, namely MusicFacebook (M.FB), MusicWebBrowser (M.WB), CameraVideo(CamV.), and Music. Second, HybridSim, a simulator equipped with pSLC technique-enabled high-density flash-based storage and a host layer, is developed based on the widely used simulator SSDSim [26]. It integrates a Markov model-based prefetcher [9][7] and CPF in the host layer. Additionally, it can emulate the storage of consumer devices used in this paper, including storage layout, mapping table, GC, and prefetching. The capacity setup of storage follows the real products [3] and the read/write performance is based on [14][15]. Third, we run the first half of the collected workloads on HybridSim to build the transition table of the Markov model-based prefetcher. Then the second half of the collected workload is used to evaluate prefetchers.

To evaluate the advantages of CPF, the following schemes are evaluated. (1) *Baseline* is the traditional read-ahead prefetcher (see Section II). (2) *Markov model-based Prefetcher (MP)* refers to the general Markov model-based prefetcher. P_T is set to 0.8 based on Section III. (3) *MP_agg* refers to the aggressive MP. P_T is set to 0.2 based on Section III. **(4) *CPF*** refers to the proposed cross-layer prefetching framework. P_{high} and P_{low} are initialized as 0.8 and 0.2, respectively.

B. Experimental Results

1) Read Performance: To evaluate the prefetching benefits, we compare the read latency of different schemes. Figure 6(a) presents the results which are normalized to *Baseline*. **For all workloads**, compared with *Baseline*, *MP*, *MP_agg*, and *CPF* can reduce read latency by 15.6%, 7.8%, and 31% on average, respectively. **For collected workloads**, *MP* achieves the smallest read performance improvement. Considering the cost of transferring data to DRAM, *MP* only prefetches the data with high accuracy. *MP_agg* has the most prefetch hits by prefetching lots of data to DRAM. Therefore, it can reduce the read latency significantly. Different from them, *CPF* prefetches data selectively with the consideration of both accuracy and coverage. Therefore, *CPF* prefetches more data to achieve 53.3% higher performance than *Baseline* and 15% higher performance than *MP*. The performance gap between SLC and DRAM makes its performance slightly lower than *MP_agg*, which is only 8.7%. **For Nexus 5 traces,** lots of data are reused in the future. This leads to *Baseline* having a good performance. As for *MP_agg* and *MP*, the read performance is worse than *Baseline*, which are 48% and 7.5% on average, respectively. This is because the read data that is required to be evicted can be read more times than the prefetched data in *MP* and *MP_agg* under these workloads. *CPF* leverages the storage capacity of SLC flash to store the most prefetched data to avoid polluting the DRAM, enabling the DRAM to store data with high accuracy. The increased storage capacity (i.e., DRAM plus SLC flash) to store prefetched data not only expands the prefetch coverage but also improves the prefetch accuracy, therefore, *CPF* still improves the read performance by 8% better than *Baseline*.

2) The Amount of Transferred Data: To figure out prefetching costs, the amount of transferred data to the host system for different schemes is collected. Figure 6(b) presents the results where three sources of transmitted data are presented.



(a) Normalized Read Latency (b) Normalized amount of transferred data (c) Realtime accuracy of CPF

Fig. 6. Experimental results on (a) Read performance; (b) Amount of transferred data; (c) Realtime accuracy.

(i) *Misses in Demand* represents the number of reads without hitting prefetched data. (ii) *Hits in SLC* represents the amount of reads that access the prefetched data in SLC. (iii) *Prefetches to DRAM* represents the amount of prefetched data to DRAM. All counts of each workload are normalized to the number of total reads of corresponding *Baseline*.

From the results, a few data are prefetched in *Baseline*. This indicates current user access patterns are too complex to catch by the traditional read-ahead prefetcher. Compared to *Baseline*, *MP_agg* prefetches a lot more data which is 2.24x on average. It causes a high prefetching cost in transferring data to DRAM. Differently, *MP* and *CPF* only prefetch 17.2% and 16.6% more data than *Baseline* on average. *CPF* has more hits in SLC than *MP* which makes it have better performance. The reason why read performance has different trends on collected workloads and Nexus 5 can also be found in Figure 6(b). *MP* and *MP_agg* reduce the number of demand misses significantly by prefetching data in collected workloads, but have a slight reduction in Nexus 5 workloads. Figure 6(c) presents the accuracy of *CPF* on a collected workload during each time slot (60 seconds). DRAM has higher prefetching accuracy to avoid the high cost of transferring data to DRAM, while SLC flash has lower prefetching accuracy to increase prefetching coverage. In conclusion, *CPF* enables the prefetcher to obtain great benefits while only transmitting a small amount of data.

VI. Conclusion

This paper revisits prefetching schemes for state-of-the-art high-density flash-based storage and proposes a cross-layer prefetching framework to trade off prefetching benefits and costs. Two components are integrated into the host system: a prefetching selector and an adaptive regulator. The former is used to prefetch data selectively. Specifically, the prefetching data with high accuracy will be prefetched to DRAM while the prefetching data with large coverage will be prefetched to SLC flash of high-density flash-based storage. The latter is used to adaptively adjust the prefetcher to make the accuracy and coverage of DRAM and SLC flash meet the expectations. With the cross-layer prefetching framework, the prefetcher can achieve both high prefetching benefits and low costs. This significantly reduces the amount of data transferred to the host while maintaining the performance gains from the prefetcher.

References

[1] L. Luo, S. Li, Y. Lv, and L. Shi, "Performance and reliability optimization for high-density flash-based hybrid ssds," Elsevier JSA, vol. 136, p. 102830, 2023.

[2] hyperstone, "How pseudo-slc mode can make 3d nand flash more reliable," 2019, https://www.hyperstone.com/en/How-pseudo-SLC-mode-can-make-3D-NAND-flash-more-reliable-2524.html.

[3] I. .Inc, "Intel 670p product," 2021, https://www.intel.com/content/www/us/en/products/sku/204109/intel-ssd-670p-series-1-0tb-m-2-80mm-pcie-3-0-x4-3d4-qlc/specifications.html.

[4] A. Ki and A. E. Knowles, "Stride prefetching for the secondary data cache," Elsevier JSA, vol. 46, no. 12, pp. 1093–1102, 2000.

[5] F. Wu, H. Xi, and C. Xu, "On the design of a new linux readahead framework," ACM SIGOPS OSR, p. 75–84, 2008.

[6] M. Li, E. Varki, and et al., "Tap: Table-based prefetching for storage caches," in USENIX FAST, 2008, pp. 81–96.

[7] A. Laga, J. Boukhobza, M. Koskas, and F. Singhoff, "Lynx: a learning linux prefetching mechanism for ssd performance model," in IEEE NVMSA, 2016, pp. 1–6.

[8] H. Wang, L. Luo, L. Shi, C. Li, C. J. Xue, Q. Zhuge, and E. H.-M. Sha, "Sfp: Smart file-aware prefetching for flash based storage systems," in ACM GLSVLSI, 2021, p. 45–50.

[9] R. Xu, X. Jin, and et al., "An efficient resource-optimized learning prefetcher for solid state drives," in IEEE DATE, 2018, pp. 273–276.

[10] X. Xu, Z. Cai, J. Liao, and Y. Ishiakwa, "Frequent access pattern-based prefetching inside of solid-state drives," in IEEE DATE, 2020, pp. 720–725.

[11] J. Li, X. Xu, Z. Cai, J. Liao, K. Li, B. Gerofi, and Y. Ishikawa, "Pattern-based prefetching with adaptive cache management inside of solid-state drives," ACM TS, vol. 18, no. 1, jan 2022.

[12] G. O. Ganfure, C. F. Wu, and et al., "Deepprefetcher: A deep learning framework for data prefetching in flash storage devices," IEEE TCAD, pp. 3311–3322, 2020.

[13] C. Chakraborttii and H. Litz, "Learning i/o access patterns to improve prefetching in ssds," ICML-PKDD, 2020.

[14] T. Kouchi, N. Kumazaki, and et al., "13.5 a 128gb 1b/cell 96-word-line-layer 3d flash memory to improve random read latency with tprog=75µs and tr=4µs," in IEEE ISSCC, 2020, pp. 226–228.

[15] A. Khakifirooz, S. Balasubrahmanyam, R. Fastow, K. H. Gaewsky, and P. Kalavade, "30.2 a 1tb 4b/cell 144-tier floating-gate 3d-nand flash memory with 40mb/s program throughput and 13.8gb/mm 2 bit density," in IEEE ISSCC, 2021, pp. 1–3.

[16] L. Shi, Y. Di, M. Zhao, C. J. Xue, K. Wu, and E. H.-M. Sha, "Exploiting process variation for write performance improvement on nand flash memory storage systems," IEEE TVLSI, vol. 24, no. 1, pp. 334–337, 2016.

[17] L. Shi, J. Li, C. J. Xue, C. Yang, and X. Zhou, "Exlru: A unified write buffer cache management for flash memory," in ACM EMSOFT, 2011, p. 339–348.

[18] C. Gao, L. Shi, C. Ji, Y. Di, K. Wu, C. J. Xue, and E. H.-M. Sha, "Exploiting parallelism for access conflict minimization in flash-based solid state drives," IEEE TCAD, vol. 37, no. 1, pp. 168–181, 2018.

[19] W. Bux and I. Iliadis, "Performance of greedy garbage collection in flash-based solid-state drives," Elsevier Perform. Eval., vol. 67, no. 11, pp. 1172–1186, 2010.

[20] S. Li, L. Luo, Y. Lv, and L. Shi, "Latency aware page migration for read performance optimization on hybrid ssds," in IEEE NVMSA, 2022, pp. 33–38.

[21] D. Zhou, W. Pan, W. Wang, and T. Xie, "I/o characteristics of smartphone applications and their implications for emmc design," in IEEE IISWC, 2015, pp. 12–21.

[22] L. Luo, D. Yu, L. Shi, C. Ding, C. Li, and E. H.-M. Sha, "Cdb: Critical data backup design for consumer devices with high-density flash based hybrid storage," in ACM/IEEE DAC, 2022, p. 391–396.

[23] J. Bhimani, J. Yang, Z. Yang, and et al., "Enhancing ssds with multi-stream: What? why? how?" in IEEE IPCCC, 2017, pp. 1–2.

[24] Y. Kim, I. Choi, J. Park, J. Lee, S. Lee, and J. Kim, "Integrated Host-SSD mapping table management for improving user experience of smartphones," in USENIX FAST, 2023, pp. 441–456.

[25] J. Axboe, Blktrace for Linux I/Os, 2006. [Online]. Available: https://git.kernel.dk/cgit/blktrace/

[26] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and C. Ren, "Exploring and exploiting the multilevel parallelism inside ssds for improved performance and endurance," IEEE TC, vol. 62, no. 6, pp. 1141–1155, 2012.