M2H: Optimizing F2FS via Multi-log Delayed Writing and Modified Segment Cleaning based on Dynamically Identified Hotness

Lihua Yang, Zhipeng Tan*, Fang Wang, Shiyun Tu, Jicheng Shao
Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System
Engineering Research Center of Data Storage Systems and Technology, Ministry of Education of China
School of Computer Science & Technology, Huazhong University of Science & Technology
Email: {lihuayang, tanzhipeng, wangfang, tushiyun, shaojc}@hust.edu.cn, *corresponding author

***Abstract*—With the widespread use of flash memory from mobile devices to large data centers, flash friendly file system (F2FS) designed for flash memory features has become popular. However, F2FS suffers from severe cleaning overhead due to its logging scheme writes. Mixed storage of data with different hotness in the file system aggravates segment cleaning. We propose multi-log delayed writing and modified segment cleaning based on dynamically identified hotness (M2H). M2H defines the hotness by the file block update distance and uses the K-means clustering to identify hotness accurately for dynamic access patterns. Based on fine-grained hotness, we design multi-log delayed writing and modify the selection and release of the victim segment. The hotness metadata cache is used to reduce overheads induced by hotness metadata management and clustering calculations. Compared with the existing strategy of F2FS, migrated blocks of segment cleaning in M2H reduce by 36.05% to 36.51% and the file system bandwidth increases by 69.52% to 70.43% cumulatively.**

***Index Terms*—F2FS, segment cleaning, hotness, K-means, multi-log delayed writing**

I. Introduction

NAND Flash has been widely used in mobile devices to large data centers in recent years. F2FS [1], a typical log-structured file system designed for Flash features, has also become popular. F2FS exhibits higher random write performance via log-structured writing and lower consistency overheads via checkpoint and roll-forward recovery. However, F2FS has excessive segment cleaning that greatly reduces its performance [2]. F2FS obtains free space through segment cleaning. There are two types of segment cleaning, foreground, and background. The foreground segment cleaning is triggered when there is not enough free space. The kernel thread wakes up periodically to perform background segment cleaning. Both types of segment cleaning conduct additional reads and writes. We do not distinguish between foreground and background segment cleaning but unified as segment cleaning in this paper. There are three steps in a cleaning process: victim selection, valid block identification and migration, and post-cleaning process. After migrating valid data, a checkpoint is made to ensure the consistency of file system.

This work is supported in part by National Key R&D Program of China NO.2018YFB1003305, NSFC No.61832020, 61821003, 61772216, National Science and Technology Major Project No.2017ZX01032-101, Fundamental Research Funds for the Central Universities.

We observe that a large number of regular files are classified as warm data but have significant differences in their hotness. Among the hot/warm/cold node and data, warm data account for the highest proportion, about 80%. Mixed storage of warm data with different hotness increases segment cleaning. Compared to the failure of all valid blocks in a segment, discrete valid blocks result in higher migration overhead and cause free space fragmentation which exacerbates segment cleaning [3]. Using the existing static semantics to express hot and cold data in F2FS is enough, but is not for warm data. The nodes are not in the scope of this paper due to relatively small number. Besides, there is difference in access hotness for the file blocks with the changes of access patterns. Therefore, we further subdivide the hotness of warm data dynamically in the F2FS existing scheme.

We propose a scheme named M2H, multi-log delayed writing and modified segment cleaning based on dynamically identified hotness. We use file block update distance (FUD) that draws on the idea of IRR (Inter-Reference Recency) [4] to define the hotness. Then identify the hotness through sampling and K-means clustering because of too many file blocks and dynamic workloads. M2H monitors changes in hotness and updates identified hotness dynamically. Then optimizes key processes of F2FS, writing, and cleaning, with fine-grained hotness of warm data. We implement multi-level logs and write warm data to different logs according to their hotness so that data with close hotness are in a segment. To avoid the performance degradation caused by random writes induced by multi-level logs, we delay submitting logs. We also delay writing warmer data to a log to reduce the write amplification caused by their frequent updates. In the three steps of segment cleaning process, we respectively make modifications. When selecting a victim, we trade off valid blocks and the age of a segment by calculating the hotness of each block in the segment. When migrating valid blocks and freeing victims, we make modifications to reduce write amplification from checkpoint. Besides, we design a hotness metadata cache for managing and calculating hotness at the file block level effectively. We deploy the M2H prototype in Linux kernel 4.13.0 and evaluate it under TPC-C [5] and YCSB [6]. Experimental results show that M2H effectively reduces segment cleaning and increases bandwidth under real workloads.

978-3-9819263-5-4/DATE21/©2021 EDAA

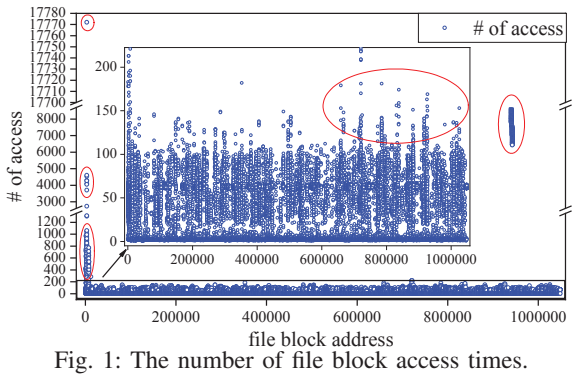

Fig. 1: The number of file block access times.

II. Motivations and Challenges

F2FS has excessive segment cleaning due to logging scheme writes. We run TPC-C and YCSB respectively and count migrated valid blocks and data written by the file system until the space utilization reaches 90%. The ratio of segment cleaning migrated blocks to normal data written in TPC-C and YCSB is 1:6.67 and 1:2.96, respectively. In other words, F2FS migrates one block for every three blocks written on average in YCSB. At best the data of different hotness are stored in different segments and some blocks that store hotter data become invalid quickly and then their segment becomes free without migration. However, the actual number of valid blocks migrated cannot be ignored. Aggressive segment cleaning generates excessive block migrations that impair user experience and shorten the storage lifetime significantly [2].

There are a lot of warm data in F2FS, but their hotness gap is relatively large. We run the tpcc-mysql [7] and count the number of writing a file block. The distribution of access times of all valid blocks in the file system is shown in Fig. 1 when the space utilization is 50%. There is a large gap in the cumulative access frequency between different file blocks while data of tpcc-mysql are classified as warm. We calculate the number of hot/warm/cold node and data blocks and the warm data and node account for 82.61% and 0.81% respectively. The warm data accounted for the majority are mixed stored with different hotness. The mixed storage of warm data with great hotness gap makes valid data discretely distributed, which leads to segment cleaning cannot select the optimal victim and has a greater migration overhead. Therefore, we aim to further subdivide the hotness of warm data on the existing scheme and optimize F2FS performance based on the fine-grained hotness.

There are two challenges of hotness identification. (1) Definition and management of hotness. We need a parameter to indicate the hotness correctly. The in-place updated file systems use the logical block address (LBA) as the key to manage the hotness of a file block. However, F2FS is updated out-of-place so that the LBA of a file block changes. (2) Accurate and dynamic identification. If the identified hotness is inaccurate and static, the optimization based on the hotness cannot work.

III. Design

A. The M2H Architecture

Fig. 2 illustrates the architecture of M2H. Data reads and writes enter the actual file system, F2FS here, from the virtual

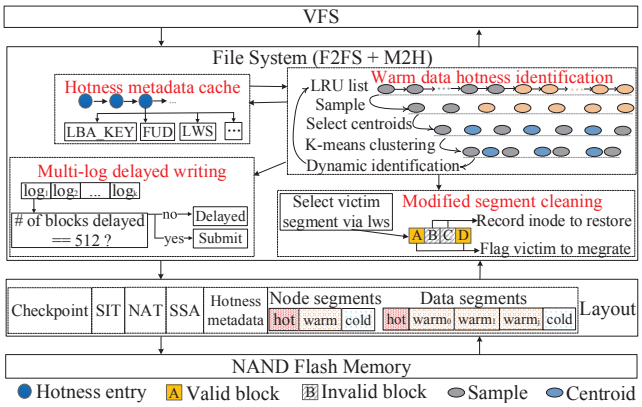

Fig. 2: The architecture of M2H.

file system (VFS). After M2H optimizes the process of data writing and cleaning, the data are written into the Flash device. The overall design of M2H can be divided into four modules: warm data hotness identification, multi-log delayed writing, modified segment cleaning, and hotness metadata cache. In the process of warm data hotness identification, sampling and clustering are deployed in the hotness metadata cache. The hotness metadata cache module is closely related to the data hotness identification module so they are introduced together in Section III-B. When the hotness identification module senses the change of hotness, it feeds back to the hotness metadata cache. The multi-log delayed writing module writes data of different hotness in batches to corresponding logs and refers to the hotness to delay the submission of warmer data. The modified segment cleaning module uses the hotness maintained by the hotness identification module to optimize the victim selection and release of segment cleaning.

B. Hotness Identification of Warm Data

1) Definition and management of hotness: Some related work for SSDs or shingled write disks inspires the definition of hotness [8]–[12]. Generally speaking, it can be divided into frequency and recency. Frequency traverses historical access information by counting the number of updates. Recency pays attention to the impact of recent updates. We believe that combining the two can define the hotness better. Inspired by the classic cache replacement algorithm, LIRS [4], we refer to its core concept, inter-reference recency, to define the hotness, file block update distance (FUD). FUD is a combination of access frequency and recency. We count it by the number of file blocks of the last two access intervals. In detail, FUD is calculated with $CWFB - LWFB$, where CWFB is the currently written file block number maintained by the file system since it works and LWFB is the last written number of the file block.

F2FS allocates a new LBA every time updates a file block. Statistics based on LBA cannot indicate the hotness correctly because the direct correspondence between LBA and file block is broken. To solve this challenge, the hotness and LBA change with the file block. When a file block is updated, the hotness information is migrated from the old LBA to the new LBA. The hotness is updated with the file block.

2) Accurate identification: F2FS distinguishes hotness by the semantics statically with a low identification overhead, such as file role (directory or regular file) and file type (multimedia file). However, a large number of regular files are classified as warm data but have significant gaps in their hotness, which requires further differentiation. Some methods empirically set multi-level thresholds to identify hotness, but cannot obtain the true hotness distribution due to an access pattern change. When it changes, the previously set thresholds will be invalid. Data clustering finds the internal correlation of data through calculation and reclassifies the data according to the similarity of attributes, which can accurately and flexibly identify hotness. We manage the hotness at file block level and a file system of 100 GB has 26,214,400 file blocks. Even if it is reduced to 1 GB, manually identifying the hotness of 262,144 blocks is quite difficult. In contrast, machine learning computes large amounts of data efficiently and adapts to dynamic workloads. ASA-FTL [10] and HAML-SSD [11] use K-means clustering to classify hot/warm/cold data in SSDs. We believe the K-means clustering can identify the hotness of warm data in F2FS.

M2H samples in the hotness metadata cache because the clustering computing affects the performance. The complexity of K-means clustering is `O(nkl)` where `n` is the number of clustering objects, `k` is the number of categories, and `l` is the number of iterations. We appropriately sample to trade off the representativeness of samples and the clustering computational overhead. We divide the linked list that stores the hotness metadata into two. The access time of the first half is relatively old and 30% of the samples is drawn, and that of the second half is relatively new and 70% is taken because the latter reflects the recent hotness distribution more clearly.

K-means algorithm needs to pre-define the number of centroids, `k`. Based on preliminary experiments, the number of categories to cluster the warm data is set to 10. When the access patterns change significantly, we suggest to calibrate the `k` offline, which is acceptable. We use layered statistics to select the initial centroids. We count the gap between the two extreme values of FUD in the sample and divide them into multiple intervals. Then calculate the number of samples in each interval and select the center of the first `k` intervals with the largest number of samples but not adjacent as the initial centroids. This method does not destroy the locality since the FUDs of a file are similar. K-means clustering monitors the changes in hotness and is performed online. It takes 1.64 to 4.15 milliseconds in current prototype to manage the hotness of a 10 GB file system.

3) Cache of hotness metadata: To reduce overheads from the hotness metadata management and clustering computing, we design a hotness metadata cache. We store all hotness metadata in Flash and partially cache them in memory, index them through the cardinality tree, and use the Least Recently Used (LRU) algorithm to replace. We use the hotness entry to store the hotness information of a file block. One block can store 512 hotness entries by calculation. During the cache replacement process, we flush the hotness metadata hierarchically.

4) Dynamic identification: We propose a dynamic hotness identification that uses the sensitivity of FUD to hotness changes. For a round of tracking, M2H tracks the hotness change when writing a file block, which calculates the new FUD and compares it with the old FUD. If the gap between the two exceeds the gap threshold, the number of changed writes is added. The number of writes tracked is also recorded until it reaches 5,000. The reason for choosing 5,000 is to trade off the change of the access pattern and a large enough sample size. Then M2H stops tracking and calculates the change ratio of the number of changed writes to the total of writes tracked. If the change ratio exceeds the pre-defined threshold, it is assumed that the current hotness distribution has changed and M2H automatically re-identifies the hotness through sampling and clustering calculation.

C. Multi-log Delayed Writing

We manage multi-level logs for fine-grained warm data since there is only a log for warm data in F2FS that causes mixed storage. However, multi-level logs induce some random writes while reducing the segment cleaning. We run TPC-C to measure the write bandwidth of single-log and multi-log mode under different file system space utilization. Single-log mode adopts native F2FS design while multi-log mode splits data into 10 categories. When the file system space utilization is less than 60%, the bandwidth of multi-log is significantly higher than that of single-log. This is because multi-log mode reduces cleaning. When it is higher than 60%, the multi-log bandwidth decreases, even lower than the single-log. This is because the free space of F2FS is exhausted and the multi-log mode triggers cleaning, however, the addresses allocated for each log are random and fragmented. Random write performance is closely related to the request size in addition to the discreteness degree of the requested address [13]. When the write request is large enough, the negative impact of scattered random writes is significantly reduced. Therefore, we delay writing multi-level logs to submit data in batches. After the data of different hotness are written into respective logs, M2H delays the submission until data in the log accumulate to a segment size. A segment in F2FS includes 512 blocks. M2H decides by whether the number of delayed blocks in a log reaches 512 as shown in Fig. 2. Besides, M2H delays writing warmer data before writing them to a log, which reduces some unnecessary writes from warmer data updates in the page cache. M2H manages a table of delay levels according to the hotness, i.e., FUD. M2H queries the table of delay levels with the FUD in the warmer data page to obtain the delay level and stores it in the hotness metadata entry.

D. Modified Segment Cleaning

We modify these three steps of the cleaning process separately. In the first step, compared with warmer data, the colder data are more valuable for selecting the victim segment. The Cost-Benefit (CB) policy in F2FS considers the number of valid blocks and the "age", but inferring the age is a rough estimate. It uses the most recent modification time of a block in a segment so the modification of a file block sets the hotness of the entire segment. The hotness of a segment in M2H is determined by the hotness of every valid block. M2H counts the average LWFB of valid blocks in a segment and uses it to represent the hotness of the segment. Then calculates the age of the segment. Finally,

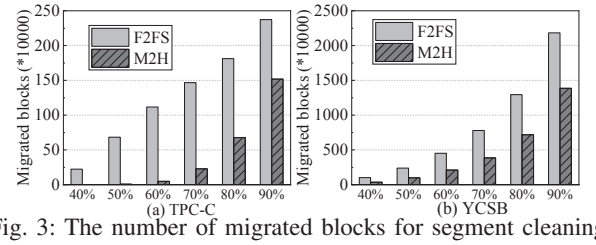

Fig. 3: The number of migrated blocks for segment cleaning.

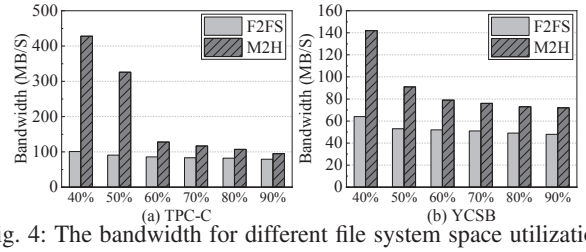

Fig. 4: The bandwidth for different file system space utilization.

referring to the CB policy that trades off the age and segment utilization, the migration cost is obtained.

In the second step, M2H adds victim flags to track the movements for valid blocks and records inode numbers for invalid blocks to reduce the write amplification induced by making checkpoints. In the third step, a checkpoint is made for consistency and pre-free segments are freed. For valid data, flush the dirty index data caused by migration via the victim flag; for invalid data, store their inode numbers in a specific location and flush. When an unexpected power outage occurs, M2H directly recovers valid data through the victim flag and finds the direct inode through the segment summary area to recover invalid data in a roll-forward recovery.

IV. Evaluation

We implement M2H by 872 lines of C code in Linux kernel 4.13.0 on a SATA SSD of Intel 730 series. M2H further distinguishes the hotness of warm data without changing the management of other file system metadata, node, and data blocks. The valid blocks migration overheads for segment cleaning under different file system space utilization are shown in Fig. 3. We monitor them by `cat status` in `/sys/kernel`. Compared with native F2FS, M2H effectively reduces segment cleaning. This is because the data in the warmer segment fails as a whole, which reduces cleaning and fragmentation even at low file system space utilization. When the space utilization reaches 90%, segment cleaning migration blocks of TPC-C and YCSB are reduced by 36.05% and 36.51%, respectively, compared to native F2FS. The segment cleaning modified by M2H also helps to select the victim segment with low migration overhead and can reduce additional reads and writes.

We measure the file system bandwidth for every 10% increase in space utilization. It uses the data write volume but not including writes caused by cleaning and write time obtained by `cat stat`. The file system bandwidth is shown in Fig. 4. The bandwidth of cumulatively written data in M2H is 69.52% and 70.43% more than that in native F2FS for TPC-C and YCSB, respectively. Compared with native F2FS, M2H has lower segment cleaning overhead. The write amplification reduced by modified segment cleaning also contributes to the increase in bandwidth. Besides, M2H delays the submission of multi-level logs of different hotness to file system. Especially when the space utilization is low, M2H has not performed segment cleaning, the bandwidth improvement is significant.

We deploy the hotness metadata cache and measure the cache hit ratio and replacement overhead. The hit ratio of the hotness metadata cache exhibits a large difference among different workloads, e.g., the hit ratio of TPC-C is better than YCSB due to stronger locality. M2H improves the hit ratio of the hotness metadata cache with acceptable memory space overhead. Since caching hotness metadata is based on the implementation of LRU, the amount of hotness metadata written induced by cache replacement is relatively small. Experimental results show that hotness metadata cache can adapt to different workloads with high hit ratio and low cache replacement overhead. Besides, we compare the CPU utilization via `/proc/stat`. The CPU utilization of M2H is 0.27% to 1.28% higher than that of native F2FS. This is acceptable compared to the performance advantages brought by M2H.

V. Conclusion

Hotness differences in large amounts of warm data exacerbate F2FS segment cleaning. We propose an optimization scheme called M2H that manages hotness at the file block level, uses the K-means clustering to analyze the distribution of hotness dynamically, and uses identified hotness to deploy multi-log delayed writing and modified segment cleaning. Besides, M2H designs a hotness metadata cache to reduce overheads induced by metadata management and clustering calculations. The experimental results indicate that M2H reduces segment cleaning overhead and outperforms native F2FS under TPC-C and YCSB.

References

[1] C. Lee, D. Sim, J. Y. Hwang, and S. Cho, "F2FS: A new file system for flash storage," in *FAST*, pp. 273–286, 2015.

[2] C. Wu, C. Ji, and C. J. Xue, "Reinforcement learning based background segment cleaning for log-structured file system on mobile devices," in *2019 IEEE ICESS*, pp. 1–8, 2019.

[3] L. Yang, F. Wang, Z. Tan, *et al.*, "Ars: Reducing f2fs fragmentation for smartphones using decision trees," in *DATE*, pp. 1061–1066, 2020.

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

[5] TPC, "TPC-C." http://www.tpc.org/tpcc/default5.asp, 2020.

[6] B. F. Cooper, A. Silberstein, E. Tam, *et al.*, "Benchmarking cloud serving systems with ycsb," in *SoCC*, pp. 143–154, 2010.

[7] Percona-Lab, "tpcc-mysql." https://github.com/Percona-Lab/tpcc-mysql, 2017.

[8] R. Stoica and A. Ailamaki, "Improving flash write performance by using update frequency," *Proceedings of the VLDB Endowment*, vol. 6, no. 9, pp. 733–744, 2013.

[9] K. Kremer and A. Brinkmann, "Fadac: a self-adapting data classifier for flash memory," in *SYSTOR*, pp. 167–178, 2019.

[10] W. Xie *et al.*, "Asa-ftl: An adaptive separation aware flash translation layer for solid state drives," *Parallel Computing*, vol. 61, pp. 3–17, 2017.

[11] B. Li, C. Deng, J. Yang, *et al.*, "Haml-ssd: A hardware accelerated hotness-aware machine learning based ssd management," in *ICCAD*, pp. 1–8, 2019.

[12] S. N. Jones, A. Amer, E. L. Miller, *et al.*, "Classifying data to reduce long-term data movement in shingled write disks," *ACM Transactions on Storage*, vol. 12, no. 1, p. 2, 2016.

[13] C. Min, K. Kim, H. Cho, *et al.*, "Sfs: random write considered harmful in solid state drives," in *FAST*, pp. 1–16, 2012.