MiDAS: Minimizing Write Amplification in Log-Structured Systems through Adaptive Group Number and Size Configuration

Seonggyun Oh, Jeeyun Kim, and Soyoung Han, *DGIST;*
Jaeho Kim, *Gyeongsang National University;* Sungjin Lee, *DGIST;*
Sam H. Noh, *Virginia Tech*

https://www.usenix.org/conference/fast24/presentation/oh

This paper is included in the Proceedings of the
22nd USENIX Conference on File and Storage Technologies.

February 27–29, 2024 • Santa Clara, CA, USA

978-1-939133-38-0

Open access to the Proceedings
of the 22nd USENIX Conference on
File and Storage Technologies
is sponsored by

MiDAS: Minimizing Write Amplification in Log-Structured Systems through Adaptive Group Number and Size Configuration

Seonggyun Oh* *DGIST*
Jeeyun Kim* *DGIST*
Soyoung Han *DGIST*
Jaeho Kim *Gyeongsang National University*
Sungjin Lee *DGIST*
Sam H. Noh *Virginia Tech*

Abstract

Log-structured systems are widely used in various applications because of its high write throughput. However, high garbage collection (GC) cost is widely regarded as the primary obstacle for its wider adoption. There have been numerous attempts to alleviate GC overhead, but with ad-hoc designs. This paper introduces MiDAS that minimizes GC overhead in a systematic and analytic manner. It employs a chain-like structure of multiple groups, automatically segregating data blocks by age. It employs analytical models, Update Interval Distribution (UID) and Markov-Chain-based Analytical Model (MCAM), to dynamically adjust the number of groups as well as their sizes according to the workload I/O patterns, thereby minimizing the movement of data blocks. Furthermore, MiDAS isolates hot blocks into a dedicated *HOT* group, where the size of *HOT* is dynamically adjusted according to the workload to minimize overall WAF. Our experiments using simulations and a proof-of-concept prototype for flash-based SSDs show that MiDAS outperforms state-of-the-art GC techniques, offering 25% lower WAF and 54% higher throughput, while consuming less memory and CPU cycles.

1 Introduction

Log-structured systems are widely used in various applications such as key-value stores (*e.g.,* LSM-trees [36, 40]), file systems (*e.g.,* F2FS [28]), and storage firmware (*e.g.,* FTLs [18, 22, 24, 31]). Log-structured systems not only provide high write throughput with fairly good latency, but are also well-suited for emerging storage media that only supports append-only writes such as NAND flash-based devices [3, 9], ZNS [6, 34, 45], and SMR [1, 2, 17] drives.

Despite such benefits, the high garbage collection (GC) cost of log-structured systems is considered its major impediment. Log-structured systems divide the storage space into fixed-size segments, each several MiB in size, while the segments themselves comprise 4KiB data blocks. A segment is the unit of space allocation and GC, and a 4KiB block is the unit of reading and writing data. Log-structured systems append new versions of data blocks to segments, leaving old ones as garbage that must be cleaned up through GC later. During GC, a victim segment with garbage blocks is identified, valid (or live) blocks are copied to another segment, and finally the new freed victim segment is returned for future writes. Relocating live blocks causes numerous extra reads and writes. A common metric to measure the impact of extra writes during GC is the *write amplification factor* (WAF), which is the ratio of the total number of blocks written to storage to the number of blocks written by the user.

Many studies have been conducted to alleviate the overhead caused by GC. These studies try to reduce WAF by employing two main techniques: victim selection [15, 23, 38] and data placement [10, 11, 27, 33, 33, 37, 42, 44, 49–51]. Despite these many efforts, existing techniques often fail to minimize WAF because of the following two limitations. The first is inaccurate prediction of block lifespan, that is, distinguishing hot and cold blocks. Hot blocks (frequently updated blocks) have short lifespan while cold blocks (infrequently updated blocks) have long lifespan. While the notion of hot and cold is well accepted, the boundary between hot and cold is relative according to workload and typically changes over time. Existing techniques cannot efficiently define such a boundary, thereby making inaccurate classification of data blocks. This results in data blocks with varying lifespans being mixed up in the same segment causing many live block copies during GC. The second is inefficient partitioning of storage space. To group blocks with similar lifespan together, existing techniques maintain groups of segments and segregate data blocks with similar lifespan to a designated group. Current state-of-the-art techniques typically work with 2–8 groups. Unfortunately, the number of groups and their sizes are decided in an ad-hoc manner resulting in suboptimal WAF.

In this paper, we propose MiDAS, a **M**igration-based **D**ata placement technique with **A**daptive group number and **S**ize configuration for log-structured systems. MiDAS employs a chain-like structure comprising multiple groups, with each group G_i linked to the subsequent group G_{i+1}. Incoming data blocks are initially written to the first group G_1 and thereafter, only valid blocks from one group G_i are moved to the next group G_{i+1} automatically segregating data blocks by age.

*These authors equally contributed to this work.

MiDAS optimizes the number of groups and their sizes according to the characteristics of the workload so that the number of valid block copies between segments is minimized. This is done by making use of analytical models, Update Interval Distribution (UID) and Markov-Chain-based Analytical Model (MCAM). By monitoring long-term trends of block updates, UID tells us how many blocks in a group remain valid and move to the next group. By leveraging the chained organization of groups in MiDAS, MCAM accurately predicts the WAF value given a specific group configuration. Using UID and MCAM, MiDAS explores a wide range of group configurations and finds one that minimizes WAF.

To further reduce WAF, MiDAS also isolates hot blocks in a group called *HOT*. To identify hot blocks, MiDAS does not rely on simple heuristics. Instead, MiDAS sends selected blocks that are soon to be invalidated to the *HOT* group, which is dynamically adjusted according to changing workloads in balance with other groups to minimize overall WAF.

While MiDAS is designed for log-structured systems, this paper specifically focuses on data placement for flash-based SSDs for evaluation. Accordingly, we have implemented MiDAS on the FTL, and all experiments are conducted at the SSD level. To understand the effectiveness of MiDAS, we conduct a simulation study using I/O traces collected from various benchmarks and real-world systems. We compare MiDAS to four state-of-the-art GC techniques: CAT [10], AutoStream [51], MiDA [37], and SepBIT [44]. Our results show that MiDAS can provide 25% lower WAF compared to the other techniques, on average. We also implement a proof-of-concept prototype of MiDAS in an SSD controller and confirm that MiDAS not only provides lower WAF and higher throughput, but exhibits better memory efficiency and consumes fewer CPU cycles than SepBIT, which is the best-performing state-of-the-art (SOTA) technique. We also include a discussion on the applicability of MiDAS to other log-structured systems.

2 Background and Related Work

2.1 Victim Selection Policies

A victim selection policy decides a victim segment with the goal of minimizing the number of live block copies during GC. Three commonly used policies are (*i*) *FIFO* [15, 38], (*ii*) *Greedy* [38, 47], and (*iii*) *Cost-Benefit* [10, 23, 38].

FIFO chooses the oldest segment as a victim. FIFO is simple to implement, but often misses opportunities to select better segments with fewer live blocks as victims.

Greedy selects the segment with the lowest utilization u (the fraction of blocks still live) as u determines the number of valid block copies. It reclaims the largest fraction of the segment space 1-u after GC. Greedy, however, often selects segments containing hot blocks, which need not be copied during GC as they will soon be invalidated [15, 38, 47].

Cost-Benefit (CB) aims to minimize GC cost by considering both the utilization and age of the segment [38] trying to avoid unnecessary copies of hot blocks. CB calculates scores for individual segments and selects one with the minimum score. A commonly used score is $\frac{u}{age \times (1-u)}$, where age represents how long the segment has been alive since its creation.

CB exhibits lower WAF than FIFO and Greedy. Despite its higher complexity, CB is widely adopted for low GC cost. The effectiveness of the victim selection policy, however, is highly correlated with the data placement policy being used.

2.2 Data Placement Policies

Various data placement policies have been proposed to further reduce WAF. The fundamental idea behind data placement is to group together data blocks with similar *invalidation time*, that is, the blocks that are likely to be invalidated at a similar time. Before writing a user data block (a user-written block) or relocating a live block from a victim segment to another, the data placement policy estimates the expected invalidation time of the block and then assigns it to an appropriate segment that is holding blocks with similar invalidation time. Then, as the blocks in the segment are all invalidated at similar times, this assists in generating dead segments, which contain only invalid blocks. Dead segments do not require any live block copies, so WAF can be significantly reduced.

The key here is in accurately estimating the invalidation time of a block. While many strategies have been suggested [21, 25, 33, 44, 49, 50], one or a combination of the following three attributes of a block is commonly used: (*i*) *update frequency* [10, 11, 27, 33, 42, 51], (*ii*) *latest update interval* [44, 50], and (*iii*) *age of the block* [33, 37, 44, 49].

2.3 Review of Prior Techniques

Here, we present four SOTA GC techniques, CAT [10], AutoStream [51], MiDA [37], and SepBIT [44], focusing on the data placement method used. CAT [10] categorizes a data block into two types, hot and cold, based on its update frequency and assigns it to either a hot or cold segment group. It dynamically changes the sizes of groups by moving live blocks over groups during GC.

AutoStream [51] designed for Multi-Streamed SSDs [16] attempts to finely categorize data blocks at the host level with support from the storage device. AutoStream counts the number of updates of data blocks on the host side and classifies them based on update frequency. Then, it sends data blocks with group IDs determined by their update frequencies to the SSD. Owing to the limit of being designed in the confines of an SSD, the number of groups is usually set to five [21].

MiDA [37] utilizes the age of a block for data placement. It creates a chain of segment groups, each of which is connected to a neighboring group. Incoming blocks are first written to Group 1, the head of the chain, with an age of 0. If it remains valid until being selected as a victim for GC, it is moved to the next group, Group 2, with an age of 1, and so on. In this way, it clusters data blocks with similar ages in the same group. There is no specific limit on the number of groups, but MiDA maintains up to eight groups by default.

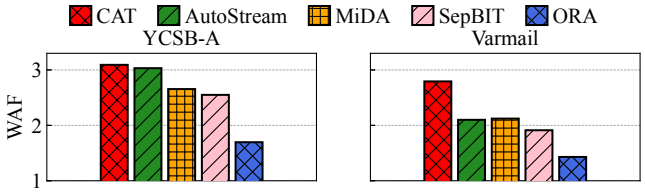


Fig. 1. WAF of ORA and existing SOTA techniques

SepBIT [44] employs both the latest update interval and age of a block. For a newly written block, it estimates the invalidation time based on its latest update interval – the time-span since the block was last written – and assigns the block to hot or cold groups. When relocating a live block during GC, SepBIT measures the age of the block and sends it to an appropriate GC group. The total number of groups, including hot, cold, and GC groups, is six. SepBIT uses a threshold-based heuristic to decide a target group where data blocks are assigned.

In summary, first, most GC techniques based on invalidation times usually group segments into 2–8 groups, and this number of groups is decided without any justification. We are aware of some prior studies that address this by dynamically changing the number of GC groups [42, 48]. For instance, Multilog estimates the update frequency of a block by employing both the LRU and Oracle algorithms [42]. If a block's update frequency falls below the average, a colder group is created, and the block is demoted to this group. Also, a comprehensive analysis of the best number of GC groups across various workloads is provided by Yadgar et al. [48]. However, these studies did not consider how the number of groups should change as the workload dynamically changes. Second, the size of the group, though dynamic in a limited way, is determined without considering what size is most appropriate to accommodate the incoming blocks destined for the group. We are not aware of any prior work that tackles this issue.

3 Motivation: Current GC Techniques

In this section, we analyze the limitations of current SOTA GC techniques through quantitative observations, which serve as motivation of our work MiDAS.

3.1 Experimental Setup

To evaluate the effect of current SOTA GC techniques, we implement the techniques within the FTL of flash-based SSDs, and make use of two benchmarks, YCSB-A [13] that runs on MySQL and Varmail of the Filebench benchmark [43]. The number of 4KiB blocks written by YCSB-A and Varmail are 4.4 billion (16.4 TiB) and 4 billion (14.9 TiB), respectively. To repeat the experiments under the same environment, the I/O traces of these benchmarks are collected and fed to a trace-driven simulator that implements the victim selection and data placement policies of the various GC techniques. The simulator models a 128GiB storage space with 64MiB segments. The experimental setup is detailed in §5.

To objectively evaluate the performance of the existing techniques, we compare them with an oracle algorithm (ORA) that minimizes WAF through offline analysis of the collected traces. Through this analysis, the invalidation time of every block is obtained, which is then used to assign blocks with similar invalidation times to the same segment group. Note, however, that deciding the optimal number of segment groups and group sizes is an NP-hard problem [29]. Thus, we perform k-means clustering [20] over the traces to find the best number of groups that accommodates data blocks with similar invalidation time. We also empirically decide the size of individual groups such that WAF is lowest. Note that ORA does not adjust the group size during victim selection because their sizes are decided a priori.

Table 1: Ranges of invalidation times for C_1–C_6 in ORA

	C_1	C_2	C_3	C_4	C_5	C_6
Range	<250K	250K–5M	5M–14M	14M–28M	28M–62M	>62M

The four SOTA techniques reviewed in §2.3 are compared against ORA. According to their original design, the number of segment groups is set to 2, 5, 8, and 6 for CAT, AutoStream, MiDA, and SepBIT, respectively. The victim selection policy is CB as it provides the best performance.

Fig. 1 shows the WAF results. Note that when reporting WAF values, throughout the paper, we consistently start from base 1 for the *y*-axis. ORA is effective with WAF being closest to 1.0 for both YCSB-A and Varmail. CAT shows the worst WAF. AutoStream, MiDA, and SepBIT, which maintain multiple groups for data placement and use more sophisticated invalidation time prediction, exhibit lower WAF. However, we still see a large gap between the existing techniques and ORA. In the following, we conduct a series of experiments to analyze where this discrepancy is coming from.

3.2 Analysis based on ORA

Accuracy of invalidation time prediction: We first evaluate how the accuracy of invalidation time prediction varies per prediction approach. Invalidation time is defined to be the number of user-written blocks, in 4KiB units, which are written between the time the block of interest is written to and the time it becomes invalid. Blocks with shorter invalidation time generally mean they are hotter.

To objectively evaluate prediction accuracy, we utilize the classifications of blocks by ORA that accurately groups data blocks depending on their actual invalidation times through offline analysis. The analysis results in ORA dividing the data blocks into six categories, C_1, C_2, ..., C_6, depending on their hotness. C_1 represents the hottest blocks (invalidation time < 250K), C6 represents the coldest (invalidation time > 62M), and the rest are in between. Table 1 lists the ranges of invalidation times for C_1, C_2,..., C_6.

Fig. 2 illustrates the accuracy of predicting invalidation times for the YCSB-A[1] workload using three different approaches: the latest update interval (employed in SepBIT),

[1]The results trend and discussions are similar for Varmail and thus, are not presented in the interest of space.

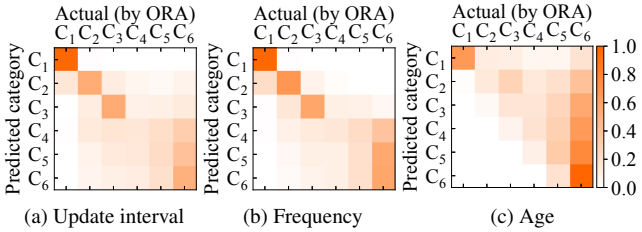


Fig. 2. Accuracy of block invalidation times for YCSB-A according to prediction techniques

Table 2: Size of each group per technique for YCBS-A (unit: segment): fixed values for ORA, while the rest are averages through workload processing

Group number	1	2	3	4	5	6	7	8
ORA	9	17	35	45	133	1,809	-	-
CAT	184	1,864	-	-	-	-	-	-
AutoStream	2	3	2	1,336	705	-	-	-
SepBIT	2	1	17	2	85	1,941	-	-
MiDA	7	79	95	126	128	117	98	1,398

update frequency (utilized in CAT and AutoStream), and the age of a block (employed in SepBIT and MiDA). In Fig. 2, we visualize the prediction accuracy of these techniques in a heatmap, comparing them to ORA. The x-axis represents the category of blocks that are decided by ORA through offline analysis. The y-axis represents the category of blocks that are predicted by each approach, also in an offline manner. More specifically, we first categorize each block as C_i based on ORA. Then, we categorize the blocks again, this time using the specific approach. For example, say there is a block A that is categorized as C_2 with ORA. Then, with the latest update interval approach, say, we observe that block A is updated at time 200K, which is in the C_1 range. Then, this block A will be a miscount that reduces the accuracy of the (C_2, C_2) zone of Fig. 2(a) and that contributes to the (C_2, C_1) zone. Thus, the intensity of the diagonal zones shows how accurate each approach is relative to ORA, while the non-diagonal zones show how much they are contributing to the inaccuracy. Ideally, if the predictions of each approach were perfect, we would only observe dark diagonal zones.

From Fig. 2, we find that the existing approaches show higher accuracy on different categories of block hotness. For latest update interval and update frequency, the prediction accuracy of hot blocks (*e.g.,* C_1), that is, those with short invalidation times, is relatively high. However, for cold blocks (*e.g.,* C_6), the prediction is much less accurate. We observe, for example, from Fig. 2(b), that blocks that actually belong to C_6 are incorrectly categorized as other C_is, even including C_2. Conversely, the age-based technique, Fig. 2(c), shows lower accuracy than the other two for hot blocks, but it excels in identifying the coldest blocks (C_6). We do see, however, that many coldest blocks are being mispredicted as other blocks ($C_1, ..., C_5$). This is likely due to the fact that many of coldest blocks are still in transit and moving towards the coldest block

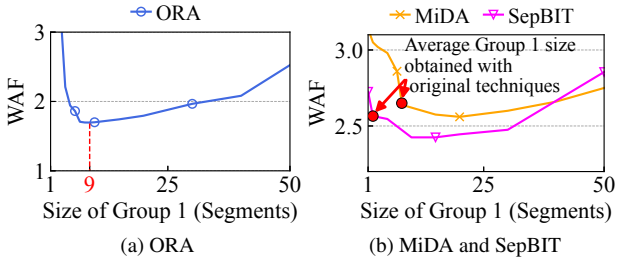


Fig. 3. Impact of group size on WAF for YCSB-A

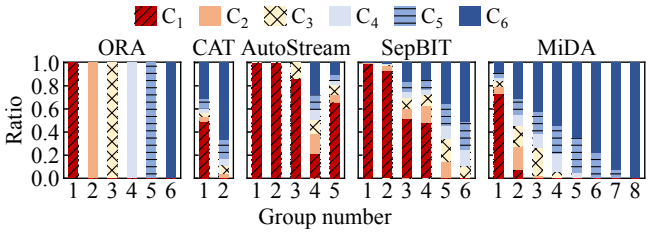


Fig. 4. Distribution of blocks over groups for YCSB-A

at the end of the experiments.

Effect of group number and size: Once we have assessed the individual block's hotness, they need to be grouped together according to their hotness. ORA, which accurately assesses hotness, through offline analysis, came up with six groups of sizes as shown in the 'ORA' column of Table 2. Fig. 3(a), where the x-axis is the size of Group 1 and the y-axis is WAF, shows how WAF changes as the ORA Group 1 size is varied. It shows that ORA chose the appropriate Group 1 size and that even with an accurate hotness assessment, incorrectly setting the group size can amplify WAF. As analysis determined that six groups show the best WAF for ORA, altering the number of groups will show similar WAF amplification.

3.3 Analysis of SOTA Techniques

As discussed, inaccurate data placement comes from two sources, inaccurate hotness predictions and inaccurate group configurations, that is, group number and size. We now attempt to quantify these issues for the four SOTA techniques.

Fig. 4 illustrates the distribution of data blocks over segment groups for the SOTA techniques. We observe that results for AutoStream based on the update frequency and SepBIT based on the update interval coincide well with the findings shown in Fig. 2, with Group 1 comprising mostly of C_1, the hot blocks. However, as shown in Table 2, the sizes for Groups 1 to 3 for these techniques are considerably smaller than those of ORA, leading to many of the hot blocks overflowing to other colder groups. We observe that for Autostream, with only five groups, the hot blocks are scattered among all the groups. Similarly, the results of the age-based MiDA technique, which generates eight groups, also coincide well with the findings shown in Fig. 2, with the coldest blocks fully filling Group 8 and forming the majority of Groups 5–7, while Group 1 shows some of its space being occupied by colder blocks. We also observe that CAT cannot perform well

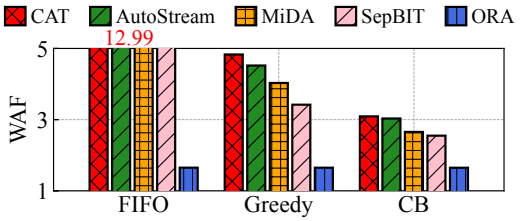


Fig. 5. WAFs of each technique by victim selection (FIFO WAF for all techniques excluding ORA is 12.99)

due to being fixed to two groups resulting in an intermix of hot and cold blocks.

To show the effect of group size on WAF, we manually control the size of Group 1 while running YCSB-A for MiDA and SepBIT. The results are shown in Fig. 3(b). Note that in our setup, as well as in typical systems, the total number of segments is fixed. Thus, as Group 1 size increases, the other groups will become smaller. The red dots in the figure show the average size of Group 1 and their original WAF values. We observe that by increasing the group size to only about 20 segments, both MiDA and SepBIT can reduce WAF by about 5.5% and 7.7%, respectively. However, further increasing the group size results in higher WAF due to the size reduction in subsequent groups. Based on these observations, the challenge becomes how to determine the number of groups to maintain and what the sizes of these groups should be such that WAF may be minimized.

Impact of victim selection: Lastly, we consider the effect of victim selection on WAF. To this end, we measure the WAF values of the five techniques with three victim selection policies: FIFO, Greedy, and CB.

Fig. 5 shows the results, from which we make two observations. First, each technique exhibits the lowest WAF when employing CB, which is a predictable outcome. This is because CB provides sufficient time for hot blocks to become invalid. Second, ORA exhibits almost the same WAF values, regardless of which victim selection policy is used. Even with FIFO, which is the simplest and where other techniques suffer, ORA can achieve low WAF. This is an interesting, yet expected result. If data blocks are perfectly distributed over different segment groups according to exact invalidation times and the group sizes are set sufficiently large, the oldest segment in the group will have the least number of valid blocks that will not be invalidated for a long time, eventually trickling down to the last group. As a result, FIFO, Greedy, and CB all behave similarly, showing almost the same WAF.

3.4 Lessons Learned: A Summary

From the results above, we make the following three key observations. *Observation #1.* Current SOTA techniques are inaccurate in predicting hotness of data. However, latest update interval and update frequency based prediction approaches tend to predict hot data relatively well, while, in contrast, age-based prediction approaches tend to predict cold data relatively well. A mix of these methods should help improve

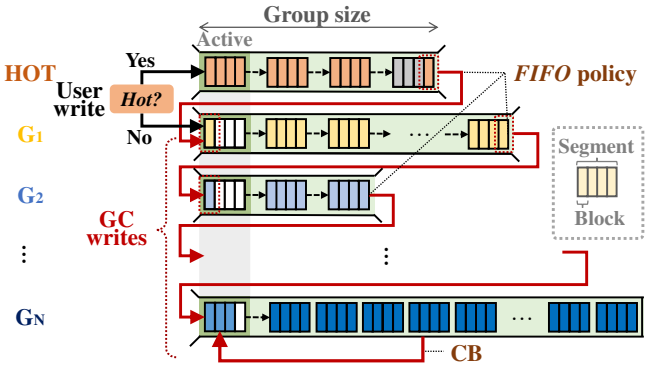


Fig. 6. Overview of MiDAS

overall predictions. *Observation #2.* The number of segment groups and their sizes have a critical impact on GC efficiency. Effort should be put into finding group number and size values that minimize WAF. *Observation #3.* FIFO is equally efficient as any elaborate policy when group sizes are properly set. With a correct data placement framework, FIFO should suffice as a victim selection policy.

4 Design of MiDAS

In this section, we present MiDAS, a technique that automatically determines the number of segment groups and their sizes to minimize WAF according to the given workload. As §3.2 illustrates, MiDAS separates the cold blocks using an age-based policy as does MiDA and separates hot blocks using update intervals as does SepBIT. In the following, we first give a high level overview of MiDAS, focusing on the relations among the key components such as the *HOT* group, UID, and MCAM. Then, in the subsequent subsections, each of these components are described in detail along with how these components interact.

4.1 Overview of MiDAS

Fig. 6 depicts the overall organization of MiDAS with $N+1$ segment groups, one *HOT* group and N cold groups, G_1, G_2, ..., G_N. From *HOT* to G_N, each segment group is linked to its next group, which creates a chain of segment groups. Upon arrival of a user-written block, the block is determined to be a hot block or not (described in §4.2). Hot blocks are directed to the *HOT* group, while others are sent to G_1, bypassing *HOT*. Every segment group, including *HOT*, has a designated size. Once the group becomes full with data blocks, a victim segment is selected from that group. Then, live blocks from this victim are migrated to the next group, G_1 for *HOT* and G_{i+1} for G_i. The freed segment is returned to the original group. The last group, G_N, does not have a next group. Thus, valid blocks from the victim are sent to G_N again, and the free segment is returned to G_N as well.

Based on the observation in §3.2, the update interval can be used as a useful means of detecting hot blocks with short invalidation times. MiDAS segregates hot blocks into the separate *HOT* group based on their update intervals. To prevent

cold blocks from being incorrectly categorized and being mixed with hot blocks in *HOT* (as seen in CAT), MiDAS sends only data blocks with short update intervals to *HOT*. At the same time, to prevent hot blocks from being sent to cold segments owing to limited *HOT* group space (as seen in SepBIT), MiDAS dynamically adjusts *HOT* to have sufficient space to accommodate the identified hot blocks.

As we have also learned from §3.2, the age-based method is effective in separating cold blocks. Therefore, MiDAS segregates data blocks with the same age in the same group, sending older blocks to the next group. Here, age is defined to be the migration count from one group to another, as is done in MiDA. All data blocks in G_i thus have the same age of $i-1$. One exception is the last group G_N, where data blocks come from G_{N-1} and from itself, that is, G_N. The ages of blocks in G_N are greater than $N-2$.

The most crucial issue in designing MiDAS is to decide the number of groups and their sizes, so that the number of valid block copies between groups is to be minimized. To make accurate decisions, MiDAS monitors long-term behaviors of block updates and creates an Update Interval Distribution (UID) model. Given a segment group with a specific size, UID tells us how many blocks in the group stay alive and move to the next group. By leveraging the chained organization of segment groups, MiDAS employs a Markov-Chain-based Analytical Model (MCAM) that accurately predicts WAF for a given group configuration. By integrating UID and MCAM, we can explore a range of group configurations, which enables us to determine the most effective combination of the number of groups and group sizes that minimizes overall WAF.

If a proper group configuration is chosen by UID and MCAM, segment groups would have sufficiently large space so that blocks are invalidated prior to eviction. This allows MiDAS to manage each segment group as a FIFO queue and to use the simple FIFO victim selection policy.

If I/O patterns of the workload are irregular and change significantly over time, our models, UID and MCAM, which rely on past history to forecast future behavior, may not make accurate decisions. Then, MiDAS simply falls back to the basic MiDA technique.

4.2 Hot Block Separation

Prior data placement techniques take various approaches to define a boundary between hot and cold. MiDA simply segregates hot from cold blocks by sending old blocks to the next group in the chain. CAT and AutoStream explicitly define a hot-cold boundary based on update frequency (*i.e.,* update counts), but with disappointing results.

SepBIT uses a more advanced approach to define a hot-cold boundary. It internally maintains a queue and pushes block numbers of every user-written block into the queue. Then, user-written blocks referenced again within the queue are sent to a designated hot group. The length of the queue is set to the average resident time, which is the time user-written blocks remain in the hot group before being removed. The queue length depends on the hot group size, which is adjusted by CB. This is an interesting and novel approach where it tries to segregate hot blocks from the rest of the blocks by adjusting the queue length based on the lifetime of the hot blocks, that is, blocks residing in the hot group.

Unfortunately, SepBIT tends to misbehave owing to how its hot group size is decided. For example, let us assume that SepBIT accurately segregates hot blocks in the hot group. Then, many dead blocks are generated from the hot group and are quickly removed due to using CB. This reduces the average resident time, which in turn shrinks the queue size. This results in only a few hot blocks being sent to the hot group, even though many more hot blocks may exist. Conversely, if many cold blocks are mistakenly sent to the hot group, the length of the queue is likely to grow because it takes longer to evict blocks from the hot group. As a result, SepBIT may assign even more cold blocks to the hot group.

MiDAS tackles the limitations that the prior techniques have by taking two unique approaches: (*i*) tight admission control to the *HOT* group and (*ii*) dynamic size adjustment of the *HOT* group. The first, tight admission control, is similar to the approach of SepBIT, but MiDAS is more conservative when deciding a block to be hot. Similar to how SepBIT promotes blocks from the queue to the hot group, MiDAS promotes, from G_1, data blocks that are soon to be invalidated to the *HOT* group. Similar to how SepBIT maintains the average resident time to set the queue length, MiDAS maintains the same value and refers it as the threshold time. The difference, though, is that this threshold time is used to decide if the G_1 to *HOT* promotion should occur or not, instead of, for setting the queue length. Specifically, MiDAS compares the update interval of the block to the threshold time, and only blocks that are updated three times[2] within the threshold time are confirmed and promoted to *HOT*. Note that the choice to use the update interval to identify hot blocks is based on §3.2.

The second unique approach of MiDAS is that the size of *HOT* adapts to the workload. As we have seen in §3, as hotness is a relative notion, segregating hot blocks from the rest is not a simple matter, which is compounded by the difficulty of setting the size of the group that will hold the hot blocks. In MiDAS, the size of *HOT* is determined in conjunction with the rest of the group such that the overall WAF is minimized. The key technical issue, then, is how to estimate the impact of the group size adjustment, including that of *HOT*, on overall WAF, without reorganizing the actual group sizes, which is costly. MiDAS can accurately predict expected WAF using MCAM and UID as explained below.

4.3 Prediction of WAF using MCAM

We now explain MCAM, a Markov-Chain-based Analytical Model, to predict the WAF value of a given group configuration. Note that Bux and Iliadis calculate WAF using a Markov

[2] A 2-bit counter is used for this, whose overhead is analyzed in §5.1.

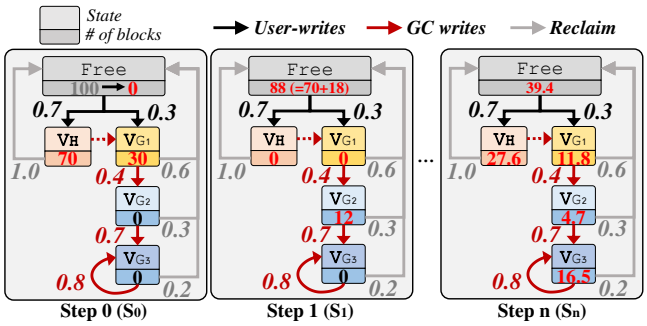


Fig. 7. Diagram of MCAM and WAF prediction process using MCAM as states transition

Chain for uniform workloads [8]. However, this prior work does not address multiple GC group scenarios needed for MiDAS, which, as depicted in Fig. 6, forms a chain of segment groups where live data blocks migrate between adjacent groups. Thus, we design MCAM to predict WAF across various group configurations, applicable to any workload pattern. If the input workload is in a steady state, MCAM is able to predict the value of WAF accurately by simulating the flow of data blocks moving over segment groups.

MCAM consists of states that blocks can be in and the transition probabilities between those states. Blocks can be in one of two states: valid (`V`) and free (`Free`). The valid state is categorized into finer states, $\texttt{V}_\texttt{H}$ and $\texttt{V}_{\texttt{G}_i}$, for $i = 1,...,N$, according to which group (*i.e., HOT* or G_i) the block is valid in. Hereafter, we use the index i to always represent the range $1,...,N$ unless otherwise stated. `Free` indicates the state of the block that is invalidated, reclaimed, and ready to use.

We now discuss transitions between states and, for clarity, refer the reader to the leftmost figure in Fig. 7 as an example with four groups (*HOT*, G_1, G_2, and G_3). For transitions from $\texttt{V}_\texttt{H}$ to $\texttt{V}_{\texttt{G}_1}$ and from $\texttt{V}_{\texttt{G}_i}$ to $\texttt{V}_{\texttt{G}_{i+1}}$, where $\texttt{V}_{\texttt{G}_{N+1}} = \texttt{V}_{\texttt{G}_N}$ (which means that $\texttt{V}_{\texttt{G}_N}$ transitions into itself), live blocks in the victim segment chosen for GC are moved from the source to the destination. The rest of the blocks are invalidated and then become free, which forms the transitions from $\texttt{V}_{\texttt{G}_i}$ to `Free`. Suppose that a fraction of valid blocks in the victim segment of G_1 is 0.4, on average. Then, the transition probability, which is the probability that blocks in one state move to another state, from $\texttt{V}_{\texttt{G}_1}$ to $\texttt{V}_{\texttt{G}_2}$ is 0.4 and the transition probability from $\texttt{V}_{\texttt{G}_1}$ to `Free` is, naturally, 0.6. In MiDAS, user-written blocks are stored in free blocks and then assigned to either *HOT* or G_1. Thus, two transitions, from `Free` to $\texttt{V}_\texttt{H}$ (0.7 in Fig. 7) and from `Free` to $\texttt{V}_{\texttt{G}_1}$ (0.3 in Fig. 7), represent the movement of user-written blocks to *HOT* and G_1, respectively. The sum of the two transition probabilities is always 1.0. All the blocks destined for *HOT* are expected to be invalidated before eviction, and thus, the transition probability from $\texttt{V}_\texttt{H}$ to `Free` is assumed to be 1.0 at all times.

Let us now discuss how WAF is predicted with MCAM using Fig. 7. We denote the progress of states as step S_k. For the moment, assume that the transition probabilities are given as in the figure. How these are obtained will be discussed in §4.4. Let us also assume that, at the initial step, S_0, we have 100 user-written blocks come in and the transition probabilities to *HOT* and G_1 are 0.7 and 0.3, respectively. Thus, we have 70 and 30 blocks in *HOT* and G_1, respectively. At the next step, S_1, 12 blocks in G_1 are moved to G_2, while the other 18 blocks are moved to `Free` by the transition probabilities from $\texttt{V}_{\texttt{G}_1}$ to $\texttt{V}_{\texttt{G}_2}$ (0.4) and from $\texttt{V}_{\texttt{G}_1}$ to `Free` (0.6). No blocks in *HOT* move to G_1; instead, all blocks (70 blocks) move to `Free` as we expect all blocks in *HOT* to be invalidated.

This transition to the next step is repeated in similar manner until the number of blocks in each group converges, whose condition is met when the number of blocks in each state no longer changes. (Note that this process has been shown to converge [7].) S_n of Fig. 7 shows an example of how the converged results would look like. Then, WAF can be predicted using the converged values by making use of the number of user writes, obtained with $\texttt{V}_\texttt{H}$ and $\texttt{V}_{\texttt{G}_1}$ in S_n as user writes are sent to either *HOT* and G_1, and GC writes, obtained by summing the number of blocks in $\texttt{V}_{\texttt{G}_2}$ and $\texttt{V}_{\texttt{G}_3}$ at S_n. Thus, for our Fig. 7 example, WAF at S_n is estimated to be 1.63 (= (27.6+11.8+4.7+16.5)/(27.6+11.8)).

To validate WAFs predicted from MCAM, we compare measured and predicted WAF values for 50 randomly created group configurations. For evaluation, we make use of the YCSB-A and Varmail benchmarks. We first run the benchmark in a prototype of MiDAS implemented in a real-world system (see §5.2 for more details) and measure WAF values for the 50 configurations. While executing the benchmark, we also collect I/O traces and measure the average transition probabilities between segment groups. Then, we replay the collected traces on an MCAM simulator configured with the transition probabilities that we measured. We find that MCAM only shows an average error rate of 0.84% and 0.7%, with a maximum error rate of 2.82% and 2.33% for YCSB-A and Varmail, respectively, which confirms that, given a group organization, MCAM produces accurate WAF predictions.

To predict WAF from MCAM, however, we must provide the transition probabilities between segment groups. In the next subsection, we show how MiDAS estimates transition probabilities and how they are supplied to MCAM at runtime.

4.4 Estimating Transition Probabilities

To estimate the transition probabilities supplied to MCAM, we introduce UID (Update Interval Distribution). Fig. 8 illustrates the UID model, where the x-axis is the update interval and the y-axis represents the probability that blocks have the corresponding update interval. That is, UID is a probability mass function (PMF) of the update intervals of user-written blocks. How UID is obtained is described in §5.1, but in the meantime, we assume we have the UID, such as Fig. 8.

UID is used to obtain the probability of whether a block remains valid after a specific period of time, which can be directly translated into a transition probability for MCAM.

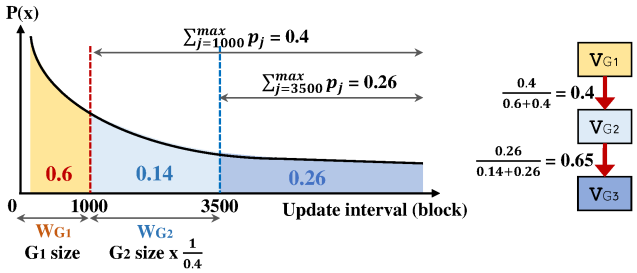


Fig. 8. Estimating transition probabilities using UID

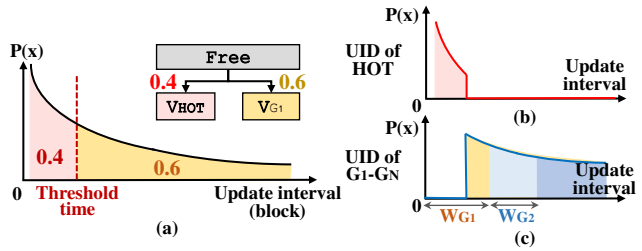


Fig. 9. Estimating transition probabilities, including *HOT*

We describe how this is done using the example in Fig. 8. Recall the unit of time in our system is defined as the number of user-written blocks. For simplicity sake, let us for now ignore *HOT* and assume only three segment groups, G_1, G_2, and G_3 each of size 1,000 exists, and that each segment is of 100 blocks. (We shall come back to *HOT* later.) User-written blocks are first sent to G_1 and then only valid blocks are moved to G_2. After 1,000 segments are written to G_1, G_1 becomes full and the victim segment at the tail of the group is selected for GC. (Recall from Fig. 6 that MiDAS uses the FIFO victim selection policy.) The number of live blocks moved to G_2 can be calculated based on UID. As the UID represents the probability of a user-written block being invalidated after a particular time interval, the sum of the probabilities of the update interval ranging from 1 to 1,000 is the probability of the blocks being invalid after 1000 writes. Let us assume, from Fig. 8, that this is 0.6, which means the transition probability from G_1 to G_2 is 0.4. Thus, out of the 100 blocks in the victim segment, 40 is transitioned to G_2.

In a similar manner, we can obtain the transition probability between G_2 and G_3. However, to fill up G_2, an additional 2,500 user-writes, that is, time steps, need to happen as only 40% of the user-written blocks are eventually sent to G_2. Once G_2 is filled with blocks, the segment at the tail of G_2 is selected as the victim. Now, let us consider how many valid blocks exist in this victim segment. To explain this, we define a new term, *waiting period* (denoted W_{G_i}), which refers to the number of user-written blocks required to fill up a specific segment group G_i. For our example, $W_{G_1} = 1000$, while $W_{G_2} = 2500$. Essentially, W_{G_i} is the elapsed time (*i.e.,* the number of user-written blocks) from when a new block comes into group G_i to when the block is evicted from the group. Now, given the UID for our workload, the sum of the probability measures for the period of W_{G_i} is the probability of the block becoming invalid. For our example, let us assume W_{G_2} is 0.14 (Fig. 8). Then, the sum of probability measures that the block remains valid is 0.26 ($= 0.4 - 0.14$). Thus, the total expected number of valid blocks expected to be moved to G_3 from the victim segment is 65 ($= 100 \times 0.65$) as the transition probability is 0.65 ($= \frac{0.26}{0.14+0.26}$).

Generalizing in this manner, the transition probability from $\mathtt{V}_{G_i}$ to $\mathtt{V}_{G_{i+1}}$, $T_{V_{G_i} \rightarrow V_{G_{i+1}}}$, is given by Eq. (1) (but not for $i = N$), where p_j is the probability for update interval j, *max* is the maximum update interval of UID, and W is the sum of the waiting periods of the groups, G_1 through G_{i-1}.

$$T_{V_{G_i} \rightarrow V_{G_{i+1}}} = \frac{\sum_{j=W+W_{G_i}}^{max} p_j}{\sum_{j=W}^{max} p_j}, \quad (1)$$

$$\text{where } W = \begin{cases} \sum_{k=1}^{i-1} W_{G_K} & \text{if } i > 1 \\ 0 & \text{otherwise.} \end{cases}$$

Transition probabilities including *HOT*: Let us now bring back $\mathtt{V}_H$ into the picture. To do that, we need to consider the transition probabilities from `Free` to $\mathtt{V}_H$ and from `Free` to $\mathtt{V}_{G_1}$. Recall from Fig. 7 that the sum of the two must be 1.0. As mentioned earlier, user-written blocks are sent to either *HOT* or $G1$, and this decision is made by referring to the threshold time. Based on this, we obtain the transition probability from `Free` to $\mathtt{V}_H$, $T_{Free \rightarrow V_H}$, by summing the probabilities of the update intervals shorter than the threshold time in UID (0.4 in Fig. 9(a)). Thus, the transition probability from `Free` to $\mathtt{V}_{G_1}$, $T_{Free \rightarrow V_{G_1}}$, is $1 - T_{Free \rightarrow V_H}$. Now, a keen reader will remember that blocks are sent to *HOT* only when it is observed that the latest update interval is less than the threshold time three times. Thus, the above explanation is not entirely correct. However, we find that not taking this into account still keeps the prediction accuracy within a maximum of 5% error. Thus, we make use of the above approximation.

Now returning back to the process that calculates the $T_{V_{G_i} \rightarrow V_{G_{i+1}}}$, we now exclude the probabilities for update intervals that are shorter than the threshold time by dividing the UID into two, UID for *HOT* and UID for G_1–G_N, as depicted in Figs. 9(b) and (c). As mentioned above, user-written blocks with update intervals shorter than the threshold time are sent to *HOT*. All blocks are also assumed to be invalidated in *HOT*. Thus, for UID of G_1–G_N, the probabilities for update intervals that are shorter than the threshold time are 0. We also need to consider the transition probability of `Free` to $\mathtt{V}_{G_1}$ when calculating W_{G_1}, resulting in the equation $W_{G_1} = \text{size of } G_1 \times \frac{1}{T_{Free \rightarrow V_{G_1}}}$. Taking the same example where G_1 size is 1,000 blocks and $T_{Free \rightarrow V_{G_1}}$ is 0.6, only 60% of the user-written blocks are sent to G_1. Thus, W_{G_1} increases to 1,667, not 1,000, as $W_{G_1} = 1{,}000 \times \frac{1}{0.6}$. For the rest of the groups, we go through the same process as before to get their W_{G_i}. Finally, the transition probabilities for $T_{V_{G_i} \rightarrow V_{G_{i+1}}}$ can be obtained using Eq. (1).

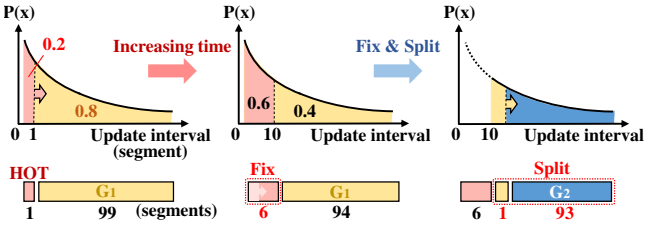


Fig. 10. Finding best group configuration with GCS

Transition probability for G_N: As the last group G_N comprises blocks of various ages as well as transitions to itself, the above analysis does not hold for G_N. Thus, to predict W_{G_N}, we make use of the analytical model proposed by Desnoyers [15][3]. This model accurately predicts WAF for techniques without data separation (*e.g.,* PageFTL [18]), where user-written blocks and GC copied blocks are placed in the same group, which is what is happening to the last group in MiDAS.

4.5 Configuring Groups with MCAM and UID

We now explain the group configuration selection (GCS) algorithm that finds the most appropriate group configuration. Given a specific group configuration, UID is able to compute transition probabilities between groups. By feeding the probabilities to MCAM, the expected WAF of the given configuration can be estimated as discussed in §4.3. Using UID and MCAM, GCS explores various group configurations to find the most appropriate one. Exploring every possible group configuration, however, is infeasible because of the huge exploration space and high computation cost.

Deciding the number of groups and their sizes: GCS takes a greedy heuristic approach to find a sufficiently good solution in reasonable time. GCS has two phases: (*i*) it roughly decides the number of groups and group sizes and then, (*ii*) fine-tunes the size of each group.

In the first phase, GCS begins with two segment groups: HOT and G_1. The primary objective of group partitioning is to ensure that for the blocks assigned to HOT, as many as possible are invalidated before eviction. The size of the HOT group needs to be carefully decided to provide sufficient time for written blocks to be invalidated before eviction. To achieve this, GCS first assigns data blocks with update intervals shorter than one segment time (the minimum unit of UID) to HOT and the rest are assigned to G_1, as depicted in Fig. 10. This figure shows an example where the sum of the probabilities within the one segment interval is 0.2. Since the unit of space allocation is a segment, GCS assigns one segment to HOT even if the suggested group size is only a few blocks smaller than a single segment.

Now that GCS has the initial sizes of HOT and G_1, it computes WAF using MCAM as explained in §4.3. GCS repeats the above step while increasing the time by one segment until a reduction in WAF is no longer observed. This is depicted in the leftmost to middle figure transition in Fig. 10. At this point, the group sizes for HOT and G_1 are determined. With HOT size fixed, the same steps are recursively repeated on G_1, that is, it splits G_1 into two groups, G_1 and G_2, and the size of G_1 increases until the observed WAF is minimized. This is depicted in the middle to rightmost figures in Fig. 10. GCS continues to split groups until no noticeable WAF reduction $> 0.5\%$ is observed over five consecutive splits.

After the number of groups and group sizes are decided, GCS goes through the second phase where the group sizes are fine-tuned. By allocating segments from the earlier groups, that is, from HOT to G_i in increasing i order, the first phase, prioritizes the earlier groups. This results in high WAF of the last group G_N as it is unlikely to get sufficient segments. To mitigate this, MiDAS reassigns segments by transferring one segment from HOT to G_N. Then, the WAF is computed. If the newly computed WAF is smaller than the old one, the movement is confirmed. Otherwise, the movement is reverted back. This is done for the remaining groups, G_i to G_N starting from $i = 1$ up to $N - 1$, until no more WAF reduction is observed.

While GCS requires moderate computation, it is invoked only when a new UID is built to adapt to changes of the workloads. Moreover, it does not affect foreground jobs (*e.g.,* write or read requests) as it is performed in the background.

Updating group configuration: The accuracy of predicting transition probabilities using UID will drop if the workload pattern changes over time. Deciding the size of a group with an incorrect UID may exacerbate overall WAF. To address this, MiDAS periodically generates a new UID and uses the new one if it provides higher accuracy. More specifically, MiDAS divides time into epochs of the same length and maintains two UIDs, one that is currently being used, which was generated in the previous epoch, and one that we generate in the current epoch. At the end of each epoch, MiDAS estimates the lowest possible WAF for the new group organization using the newly created UID and compares it to the WAF of the existing group organization with the old UID. If the WAF is not reduced by more than a certain threshold MiDAS continues to use the UID. However, if the difference exceeds the threshold, MiDAS adopts the new group organization based on the new UID. Currently, the threshold is set to 5%. Note that for the first epoch where no information about the workload is available, MiDAS simply employs the age-based MiDA policy with the CB victim selection policy.

The length of the epoch may have an impact on UID accuracy. A short epoch enables quick adaptation to changes in workload patterns but can lead to an accuracy drop due to the small amount of information. A lengthy epoch, on the other hand, may increase the accuracy of UID, but will make MiDAS less sensitive to workload changes. We experimentally determine the epoch to be four times the storage capacity. We evaluate the impact of the epoch length in §6.3.

Handling irregular I/O patterns: While MiDAS provides high accuracy in predicting transition probabilities using UID

[3]The specific model is presented in our supplemental material [41].

when workloads have regular I/O patterns, in reality, not all I/O patterns are regular. In many of the traces that we analyzed, we observe workloads with high I/O fluctuation and sudden unexpected I/O pattern changes over time. In such cases, MiDAS may fail to predict the future behavior of the workload using UID, resulting in high WAF. Thus, MiDAS takes a different approach, which we elaborate below.

First, MiDAS needs to check for irregularities. This is done by regularly checking how much the predicted and actual transition probabilities differ. This is easy to do as we have the predicted transition probabilities derived from UID and the actual transition probabilities can be obtained by keeping track of the number of valid blocks moved from one group to another. Once a high error rate is detected between groups, say between G_k and G_{k+1}, MiDAS gives up on adjusting group sizes for all groups beyond G_k and simply merges the groups from G_k to G_N into G_k. This is because the low accuracy between G_k and G_{k+1} will have a cascading impact on the accuracy of the subsequent groups. Then, MiDAS falls back to the simple age-based MiDA technique as we did when no workload information was available. This is maintained until an up-to-date UID is generated at the next epoch and can be used to find a new group configuration. Currently, this fallback mechanism is invoked when the error rate between the predicted and actual transition probabilities exceeds 10%.

5 Implementation and Experimental Setup

In this section, we discuss some implementation details including methods used to optimize memory. We then detail our experimental setup.

5.1 Implementation and Resource Overhead

In MiDAS, a chain of the cold groups, each organized as a simple FIFO queue, is managed using a linked list that consumes little memory. When the group configuration is changed, the segments containing data blocks do not physically move across the groups. Instead, only pointers that point to the physical location of each segment in the queue are moved. This allows for simple adjustment of group configurations without any data copy overhead. For example, when merging two groups, the pointers are relocated to one of the queues. Conversely, when dividing a single group into two, MiDAS initially creates a new queue and a free segment for the newly created group. Subsequently, the pointers from the segments in the original group are moved upon the activation of GC in that group.

MCAM requires moderate CPU cycles (we will discuss this in §6.4) but only needs to keep a few parameters (*i.e.,* transition probabilities and block counts) that require minimal memory. For *HOT* and UID, however, MiDAS has to keep various data structures. We discuss optimizations that we conduct to reduce memory overhead.

Hot filter: Recall that MiDAS promotes a data block to *HOT* when its update interval is found to be within the threshold time three times. To manage this, MiDAS maintains a 2-bit hot filter per block in memory. We find that this requires less than 60MiB per 1TiB storage.

Constructing UID: To construct UID, MiDAS uses two key data structures: a timestamp table and an interval count table. The timestamp table records timestamps of block updates to compute the update intervals of data blocks. To save memory space, instead of keeping track of all data blocks, we sample only a subset for timestamp monitoring, with a sampling rate of 0.01 (one in every 100 blocks). This reduces the timestamp table size to 10.3MiB per 1TiB storage. The interval count table keeps track of the number of blocks for specific update intervals. To reduce the size of the interval count table, we use a coarse-grained update interval unit of 16K blocks rather than a block unit. Blocks with update intervals falling within the range of [1, 16K blocks] are considered to have the same update interval. In this way, the interval count table reduces to 256KiB in size per 1TiB storage. The accuracy degradation by this optimization is less than 3%.

Table 3: Characteristics of each workload

Workloads	FIO-H & -M	Varmail	YCSB-A & -F	TPC-C	Alibaba	Exchange
Notation	F-H & F-M	V	Y-A & Y-F	T-C	Ali	Ex
Write size (TiB)	15.1	14.9	16.4 & 18.0	16.1	Up to 2.8	Up to 2.9
Device size (GiB)	128				40-200	40
Request distribution	Zipfian (1.0 & 0.8)	Zipfian			-	-

5.2 Experimental Setup

We carry out experiments using both trace-driven simulations and a real SSD prototype. An existing SSD simulator [12], with block I/O traces collected from various environments, is used to quickly evaluate key performance metrics (*e.g.,* WAF) of the GC techniques. We also use a real SSD prototype, where MiDAS is implemented within the FTL, to measure I/O performance and the overheads associated with the CPU and memory for system execution. Our SSD prototype is equipped with a quad-core ARM Cortex-A53 and 256MiB DRAM for indexing memory. The SSD platform employs a 256GiB custom flash array card with 16KiB flash page size and 128 pages per block and 8×8 channel. Unless otherwise stated, the over-provisioning ratio is set to 7.3% as this is typical in commercial systems [26, 39]. For both the simulator and SSD prototype, we implement the CAT, AutoStream, MiDA, and SepBIT SOTA GC techniques.

The following write-intensive workloads are used for our evaluations: FIO workloads [4] whose data access pattern follows Zipf distribution with theta values of 1.0 (denoted FIO-H) and 0.8 (denoted FIO-M), where the former and latter, respectively, represent workloads with skewed and relatively uniform data access patterns; Varmail, the most write-intensive workload in the Filebench benchmark [43]; YCSB-A and -F, the write-intensive workloads of the YCSB benchmark [13]; TPC-C [14] running on MySQL [46]; 9 Exchange traces from Microsoft Enterprise Traces [35]; and 25 write-

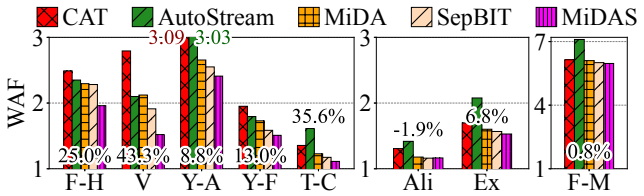


Fig. 11. Overall WAF of each technique (the numbers represent the improvement on WAF by MiDAS relative to SepBIT)

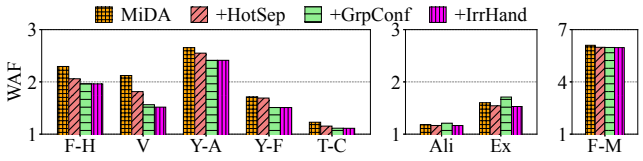


Fig. 12. Impact of individual components of MiDAS

intensive traces from the Alibaba Cloud I/O traces [30], whose LBA ranges are smaller than 256GB. Note that the traces of Exchange and Alibaba, individually, are relatively small. For these two, each trace is run individually and the average results are reported. The workload summary and details are shown in Table 3, and we use the abbreviated notations there to denote the workloads in reporting the results. Before running the workloads, we fill up 92.7% of the SSD space with data to make the SSD quickly reach a steady-state condition that invokes GC regularly. GC is triggered when free space drops below 0.1% of the total SSD capacity.

6 Experimental Results

6.1 Comparison of GC Efficiency

We first evaluate GC efficiency by comparing the WAFs of the various techniques. All experiments to measure WAFs are conducted using the trace-driven simulator. All techniques, except for MiDAS, adopt CB, the most efficient victim selection policy. Fig. 11 shows WAF for each technique. The numbers on or above each workload bar represent the improvement on WAF (%) by MiDAS relative to SepBIT, which is the best performing SOTA technique.

With FIO-H, where data access is highly skewed, a noticeable WAF reduction is observed. This is because many hot blocks are filtered out by *HOT*, which reduces the number of valid block copies during GC. By setting the hot boundary precisely and adjusting the number of cold groups and their sizes, MiDAS outperforms the SOTA techniques. (A breakdown of each contributing factor is given and discussed in more detail in §6.2.) On the other hand, with FIO-M, where almost all of the data blocks have similar update intervals with little variations, WAF reduction by hot-cold separation and group size adjustment is limited.

For Varmail, YCSB-A and -F, and TPC-C, MiDAS reduces WAF by 8.8–43.3% compared to SepBIT. These workloads not only have numerous hot blocks but also a significant number of warm blocks. MiDAS captures these characteristics with UID and then, changes the group configuration accordingly. As a result, MiDAS ensures that the warm blocks are invalidated before eviction, significantly reducing unnecessary data block copies (see §6.2).

In case of the Alibaba workloads, MiDAS and SepBIT show similar WAF, with MiDAS showing worse WAF on average. Recall that the Alibaba workloads are relatively short, showing write sizes of 2.23 epochs on average, so most of the traces end before the obtained UID can reap its benefits. Moreover, Alibaba workloads show irregular I/O patterns that increase the inaccuracy of predictions. This leads MiDAS to incorrect group configurations or to simply fall back to the MiDA technique. Thus, we see that WAF of MiDAS and MiDA are also very similar. Exchange also is composed of 9 small workloads and we observe similar irregular I/O patterns. Still, MiDAS is able to reduce WAF by 6.8% compared to SepBIT.

Overall, for the workloads we considered, MiDAS reduced WAF by an average of 25%. If we exclude the workloads with irregular patterns, Alibaba and Exchange, and the one with relatively uniform data access, FIO-M, the reduction on average is 34.7%.

6.2 Impact of Each Component of MiDAS

We analyze the impact of the individual design components of MiDAS, that is, (*i*) hot block separation (§4.2), (*ii*) group configuration (§4.3–§4.4), and (*iii*) irregular I/O pattern handling (§4.5). To observe the effect of adding each component, we start off with MiDA, the basic design that MiDAS takes from. Then, we add each design component denoted, +HotSep, +GrpConf, +IrrHand, respectively, one after the other, observing the performance as each component is added.

Fig. 12 shows how each component improves (or sometimes worsens) WAF. We distinguish the workloads into two groups, Group 1, those more typical workloads with some skewness and regularity in data access and the other, Group 2, those whose I/O patterns are largely irregular (Exchange, Alibaba) and that with low skewness in access (FIO-M), as they show distinctly different influence.

Those in Group 1 show +HotSep and +GrpConf, individually, bringing about considerable gains. As explained in §6.1, +HotSep improves the performance of FIO-H by separating hot blocks. +GrpConf is effective in Varmail, YCSB-A and -F, and TPC-C by organizing groups to give sufficient space to warm blocks. In contrast, +IrrHand shows minimal, if any, benefits. We see slight gains with Varmail and TPC-C, where small irregular patterns are detected, which is reflected in Table 4 as will be explained below.

The results for Group 2 are quite different. We see that the effect of +HotSep is smaller than for Group 1. In case of FIO-M, the impact of each component is limited as the block access is less skewed and has no irregular patterns. Thus, recall from Fig. 11 that, for FIO-M, all the techniques, except Autostream, showed similar WAF. For Alibaba and Exchange

Table 4: Accuracy of UID in predicting transition probabilities

Workloads	F-H	V	Y-A	Y-F	T-C	Ali	Ex	F-M
Avg. error (%)	1.82	6.90	2.33	1.78	4.81	**11.44**	**8.16**	1.33
Avg. error (%) w/ +IrrHand	1.82	1.97	2.33	1.78	2.38	4.67	3.36	1.33

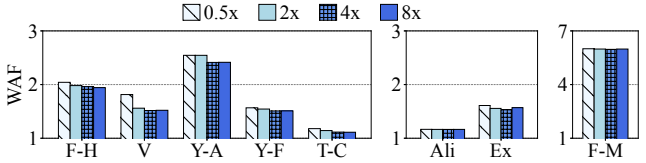


Fig. 13. Impact of length of epoch on generating UID

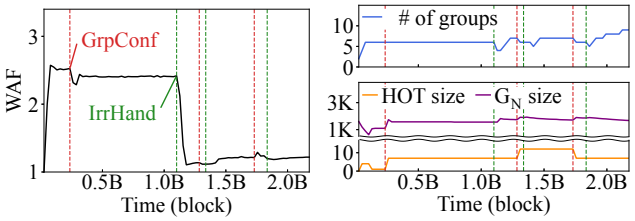
+GrpConf has a profound negative effect due to their irregular I/O patterns that make finding suitable group configurations difficult. The first row in Table 4 lists the average errors between UID predicted and actual transition probabilities. The second row is the same error but when +IrrHand is applied. We see these values coming down for Varmail, TPC-C, Alibaba, and Exchange. Before +IrrHand is applied, MiDAS is making inaccurate decisions in configuring groups based on these erroneous predictions. Hence, when +GrpConf is applied, WAF increases for Alibaba and Exhange. For Varmail and TCP-C, the inaccuracy is not as serious, so we see improvements after applying +GrpConf. We also see that +IrrHand mitigates the negative effect of irregular I/O patterns.

6.3 Miscellaneous Results

Impact of epoch length: We consider epoch lengths of 0.5×, 2×, 4×, and 8× of the storage capacity (128GiB) and observe the WAFs. Recall that all the earlier experiments were performed with an epoch length of 4×128GiB. As shown in Fig. 13, workloads are largely insensitive to epoch lengths of 4× and above with the exception of Exchange whose irregular patterns have strong influence. In contrast, shortening the length negatively affects most workloads.

Adapting capability: We evaluate how well MiDAS responds to changes in workload patterns. To see this, we run YCSB-A from 0 to 1.1 billion (B) time and then switch to TPC-C until the end of the experiment. While running, we measure WAF, the number of groups, and *HOT* and G_N sizes. Fig. 14(a) depicts how WAF changes over time. Adapting group configuration (+GrpConf) occurs six times during the entire process, while irregular pattern handling (+IrrHand) is invoked five times in each epoch. The points where MiDAS adopts GrpConf and IrrHand are highlighted by the dotted lines. The group configuration set at 0.25B remains unchanged until 1.1B due to stable transition probabilities in the YCSB-A workload (see Table 4). Once TPC-C begins at 1.1B, the I/O pattern changes dramatically, with IrrHand merging groups with high error rates to optimize WAF.

Fig. 14(b) shows how the number of groups, the *HOT* and G_N sizes evolve. We observe that when IrrHand is invoked, the size of G_N changes, while *HOT* remains unchanged due to low error rates. We conclude that MiDAS is effectively



(a) WAF progress

(b) # of groups and the size of *HOT* and G_N

Fig. 14. MiDAS adapting to workload change

accommodating the needed changes to the groups (number and size) according to the changes in the workload.

Comparison with ORA: We pointed out the limitations of the existing techniques in data placement and group size decisions by comparing them with ORA in Fig. 4 and Table 2, respectively. We now compare MiDAS with ORA to show how efficiently MiDAS addresses these limitations. Fig. 15(a) shows the distribution of data blocks assigned to groups for YCSB-A and the group size of each group (numbers on each bar). The figure also displays the ratios of block categories, C_1, ... C_6, decided by ORA. Note that the number of groups in MiDAS is the same as that of ORA and that this was reached through adjustments. We also observe that most of the hot blocks categorized as C_1 are assigned to *HOT*. MiDAS also provides sufficient space to *HOT* so that hot blocks can be invalidated before being evicted to G_1. However, we observe that, though much more accurate than the other techniques, data blocks in cold categories (*i.e.,* C_4–C_6) account for non-trivial portions in G_1–G_2. This is owing to the age-based migration policy of MiDAS that writes data blocks to the former groups and then moves live blocks to the subsequent groups. Despite such errors, age-based migration enables us to efficiently segregate cold blocks in the latter groups (G_3–G_6), helping reduce overall WAF.

6.4 Experiments on SSD Prototype

To evaluate throughput and CPU utilization that cannot be measured from the trace-driven simulator, we implement a proof-of-concept prototype of MiDAS in a real-world SSD and carry out a set of experiments. We implement PageFTL, MiDA, and SepBIT in the SSD platform. Six workloads, Varmail, YCSB-A, YCSB-F, TPC-C, Alibaba and Exchange, are used for the throughput experiments. For Alibaba and Exchange, we report the average throughput of the subtraces.

As shown in Fig. 15(b), MiDAS achieves 2.55×, 1.24×, 1.15× higher throughput, respectively, than PageFTL, MiDA, and SepBIT, on average. Notably, for Varmail, MiDAS, respectively, achieves 3.2×, 1.4× and 1.3× gains. This is because MiDAS can significantly reduce writes by GC compared to other techniques, evidenced by the WAF values of 4.26, 2.12, 1.91, and 1.52 for PageFTL, MiDA, SepBIT, and MiDAS, respectively. These are in line with the simulation results. Moreover, for Alibaba workloads, MiDAS improves

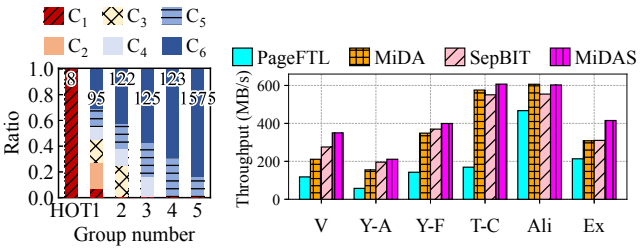


Fig. 15. (a) MiDAS block distribution, (b) Throughput result

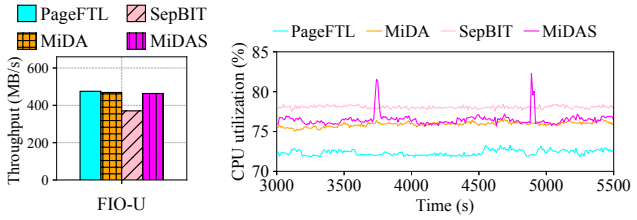


Fig. 16. Results from our SSD prototype for FIO-U

the throughput by 8.7% compared to SepBIT despite MiDAS showing higher WAF than SepBIT.

We also conduct experiments to measure the CPU overhead. To remove the GC impact, we use the FIO workload with uniform distribution (FIO-U) configured so that the WAFs of all the techniques become almost identical (1.2) and allow only 60% of the entire device to be composed of valid blocks. We use this configuration to assess I/O throughput when the techniques handle ordinary user requests without being impacted by GC. Fig. 16(a) shows that all the techniques, except for SepBIT, exhibit similar throughput as they have the same WAF. SepBIT shows the lowest throughput among the techniques. This is because it incurs high CPU utilization to maintain the queue as well as to lookup the queue for every user-written block in order to detect hot blocks. Note that the negative impact on CPU utilization is particularly significant in FIO-U where user-written blocks are dominant. For garbage-collected blocks, SepBIT does not need to look up the queue, so CPU overhead is much lower when the GC is frequently triggered.

Fig. 16(b) shows the CPU utilization of FIO-U over time. MiDAS shows, on average, only 4.2% and 0.6% higher CPU utilization compared to PageFTL and MiDA, respectively. MiDAS is designed to require only a few CPU cycles in common paths such as reads, writes, and GC. However, we notice sharp increases in CPU utilization at around 3740s and 4890s. This is where MiDAS starts to run MCAM to find the best group configuration. However, its impact on performance is minimal as this is run in the background. As shown in Fig. 16(a), MiDAS provides similar throughput as PageFTL and MiDA.

7 Discussion

In the current stage, our work is limited to a case study for flash-based SSDs. However, it can be adapted to other log-structured systems. For instance, for ZNS SSDs, MiDAS can be integrated into a zoned storage backend (*e.g.,* ZenFS [6]) to reduce the host's GC overhead, similar to SepBIT [44]. The segment size in MiDAS is not fixed and thus can be adjusted to match the characteristics of ZNS SSDs.

However, there could be several hurdles when adapting to other log-structured systems, such as the LSM-tree. LSM-trees have chain-like structures that merge-sort data through progressively larger layers [36, 40]. MiDAS's structure is similar, where live blocks migrate to subsequent groups via GC, and its group configuration could be integrated into LSM-tree level design. However, a new challenge arises because LSM-trees keep objects in a sorted order, conflicting with MiDAS's assumption that blocks in a segment share similar age. Additionally, the significantly larger key range of LSM-tree objects compared to the LBA range leads to an expanded timestamp table, thereby causing an increased memory footprint. We plan to address these problems as part of our future work.

Many researchers have discussed write amplification reduction techniques in log-structured systems like ZNS and LSM-tree [5, 19, 32, 52]. However, research that dynamically applies group configuration based on the workload pattern in log-structured systems other than flash-based SSDs has not been explored in great depth. MiDAS can provide useful insights on how to effectively apply group configurations within general log-structured systems.

8 Conclusion

In this paper, we presented MiDAS, a systematic solution to mitigate GC overhead for log-structured systems. MiDAS employs a chain-like structure to organize data by age and minimize data movement between groups using analytical models, UID and MCAM. It also isolates hot blocks within a designated hot group and dynamically adjusts its size against cold groups in a manner that minimizes overall GC costs. Our experiments demonstrated that MiDAS outperforms existing techniques, achieving 25% reduction in WAF and 54% higher throughput, on average for the workloads considered, all while being memory-efficient and consuming fewer CPU cycles.

Acknowledgments

We thank our shepherd, professor Ming-Chang Yang, and the anonymous reviewers for all their helpful comments. This work was supported by SNU-SK Hynix Solution Research Center (S3RC), the National Research Foundation of Korea (NRF-2018R1A5A1060031), the MOTIE (Ministry of Trade, Industry & Energy) (1415181081), KSRC (Korea Semiconductor Research Consortium) (20019402), and NSF grant (2312785). (Corresponding author: Sungjin Lee)

References

[1] Abutalib Aghayev and Peter Desnoyers. Skylight—A window on shingled disk operation. In *Proceedings of the USENIX Conference on File and Storage Technologies*, pages 135–149, 2015.

[2] Abutalib Aghayev, Theodore Ts'o, Garth Gibson, and Peter Desnoyers. Evolving Ext4 for Shingled Disks. In *Proceedings of the USENIX Conference on File and Storage Technologies*, pages 105–120, 2017.

[3] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark S. Manasse, and Rina Panigrahy. Design tradeoffs for SSD performance. In *Proceedings of the USENIX Annual Technical Conference*, pages 57–70, 2008.

[4] Jens Axboe. FIO: Flexible I/O Tester Synthetic Benchmark. https://github.com/axboe/fio, 2023.

[5] Oana Balmau, Diego Didona, Rachid Guerraoui, Willy Zwaenepoel, Huapeng Yuan, Aashray Arora, Karan Gupta, and Pavan Konka. TRIAD: Creating Synergies Between Memory, Disk and Log in Log Structured Key-Value Stores. In *Proceedings of the USENIX Annual Technical Conference*, pages 363–375, 2017.

[6] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R. Ganger, and George Amvrosiadis. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs. In *Proceedings of the USENIX Annual Technical Conference*, pages 689–703, 2021.

[7] Pierre Brémaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. Springer Science & Business Media, 2001.

[8] Werner Bux and Ilias Iliadis. Performance of Greedy Garbage Collection in Flash-Based Solid-State Drives. *Performance Evaluation*, 67(11):1172–1186, 2010.

[9] Feng Chen, David A. Koufaty, and Xiaodong Zhang. Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives. In *Proceedings of the Eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, pages 181–192, 2009.

[10] Mei-Ling Chiang and Ruei-Chuan Chang. Cleaning Policies in Mobile Computers Using Flash Memory. *Journal of Systems and Software*, 48(3):213–231, 1999.

[11] Mei-Ling Chiang, Paul C.H. Lee, and Ruei-Chuan Chang. Using Data Clustering to Improve Cleaning Performance for Flash Memory. *Software: Practice and Experience*, 29(3):267–290, 1999.

[12] Chanwoo Chung, Jinhyung Koo, Junsu Im, Arvind, and Sungjin Lee. Lightstore: Software-defined Network-attached Key-value Drives. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 939–953, 2019.

[13] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the ACM symposium on Cloud Computing*, pages 143–154, 2010.

[14] The Transaction Processing Council. TPC-C Benchmark. https://www.tpc.org/tpcc, 2021.

[15] Peter Desnoyers. Analytic Models of SSD Write Performance. *ACM Transactions on Storage*, 10(2):1–10, 2014.

[16] Samsung Electronics. Multi-Stream Write SSD. *Flash Memory Summit*, 2016.

[17] Garth Gibson and Greg Ganger. Principles of Operation for Shingled Disk Devices. In *Proceedings of the USENIX Workshop on Hot Topics in Storage and File Systems*, 2011.

[18] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 229–240, 2009.

[19] Kyuhwa Han, Hyunho Gwak, Dongkun Shin, and Jooyoung Hwang. ZNS+: Advanced Zoned Namespace Interface for Supporting In-Storage Zone Compaction. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, pages 147–162, 2021.

[20] John A. Hartigan and Manchek A. Wong. Algorithm AS 136: A K-means Clustering Algorithm. *Applied Statistics*, 28(1):100–108, 1979.

[21] Jeong-Uk Kang, Jeeseok Hyun, Hyunjoo Maeng, and Sangyeun Cho. The Multi-streamed Solid-State Drive. In *Proceedings of the USENIX Workshop on Hot Topics in Storage and File Systems*, 2014.

[22] Woon-Hak Kang, Sang-Won Lee, Bongki Moon, Gi-Hwan Oh, and Changwoo Min. X-FTL: Transactional FTL for SQLite Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 97–108, 2013.

[23] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda. A Flash-Memory Based File System. In *Proceedings of the USENIX Annual Technical Conference*, pages 155–164, 1995.

[24] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. A Space-efficient Flash Translation Layer for CompactFlash Systems. *IEEE Transactions on Consumer Electronics*, 48(2):366–375, 2002.

[25] Taejin Kim, Duwon Hong, Sangwook Shane Hahn, Myoungjun Chun, Sungjin Lee, Jooyoung Hwang, Jongyoul Lee, and Jihong Kim. Fully Automatic Stream Management for Multi-Streamed SSDs Using Program Contexts. In *Proceedings of the USENIX Conference on File and Storage Technologies*, pages 295–308, 2019.

[26] Kingston. Understanding SSD Over-provisioning (OP). https://www.kingston.com/en/blog/pc-performance/overprovisioning, 2019.

[27] Kevin Kremer and André Brinkmann. FADaC: A Self-Adapting Data Classifier for Flash Memory. In *Proceedings of the ACM International Conference on Systems and Storage*, pages 167–178, 2019.

[28] Changman Lee, Dongho Sim, Joo Young Hwang, and Sangyeun Cho. F2FS: A New File System for Flash Storage. In *Proceedings of the USENIX Conference on File and Storage Technologies*, pages 273–286, 2015.

[29] Jan Van Leeuwen. *Handbook of theoretical computer science (vol. A) algorithms and complexity*. Mit Press, 1991.

[30] Jinhong Li, Qiuping Wang, Patrick P.C. Lee, and Chao Shi. An In-Depth Analysis of Cloud Block Storage Workloads in Large-Scale Production. In *Proceedings of IEEE International Symposium on Workload Characterization*, pages 37–47, 2020.

[31] Seung-Ho Lim, Hyun Jin Choi, and Kyu Ho Park. Journal Remap-based FTL for Journaling File System with Flash Memory. In *Proceedings of the International Conference on High Performance Computing and Communications*, pages 192–203, 2007.

[32] Lanyue Lu, Thanumalayan Sankaranarayana Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. WiscKey: Separating Keys from Values in SSD-conscious Storage. In *Proceedings of the USENIX Conference on File and Storage Technologies*, pages 133–148, 2016.

[33] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. SFS: Random Write Considered Harmful in Solid State Drives. In *Proceedings of the USENIX Conference on File and Storage Technologies*, pages 1–16, 2012.

[34] Jaehong Min, Chenxingyu Zhao, Ming Liu, and Arvind Krishnamurthy. eZNS: An Elastic Zoned Namespace for Commodity ZNS SSDs. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation*, pages 461–477, 2023.

[35] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. Migrating Server Storage to SSDs: Analysis of Tradeoffs. In *Proceedings of the ACM European Conference on Computer Systems*, page 145–158, 2009.

[36] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. The Log-structured Merge-tree (LSM-tree). *Acta Informatica*, 33(4):351–385, 1996.

[37] Hyunseung Park, Eunjae Lee, Jaeho Kim, and Sam H. Noh. Lightweight Data Lifetime Classification Using Migration Counts to Improve Performance and Lifetime of Flash-Based SSDs. In *Proceedings of the ACM SIGOPS Asia-Pacific Workshop on Systems*, pages 25–33, 2021.

[38] Mendel Rosenblum and John K. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM Transactions on Computer Systems*, 10(1):26–52, 1992.

[39] Samsung Electronics. Over-provisioning Benefits for Samsung Data Center SSDs. https://download.semiconductor.samsung.com/resources/white-paper/S190311-SAMSUNG-Memory-Over-Provisioning-White-paper.pdf, 2019.

[40] Russell Sears and Raghu Ramakrishnan. bLSM: A General Purpose Log Structured Merge Tree. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 217–228, 2012.

[41] Seonggyun Oh, Jeeyun Kim, Soyoung Han, Jaeho Kim, Sungjin Lee, and Sam H. Noh. Supplemental Material of MiDAS. https://github.com/dgist-datalab/MiDAS/blob/main/MiDAS_supplemental_material.pdf, 2024.

[42] Radu Stoica and Anastasia Ailamaki. Improving Flash Write Performance by Using Update Frequency. *VLDB Endowment*, 6(9):733–744, 2013.

[43] Vasily Tarasov, Erez Zadok, and Spencer Shepler. Filebench: A Flexible Framework for File System Benchmarking. *The USENIX Magazine*, 41(1):6–12, 2016.

[44] Qiuping Wang, Jinhong Li, Patrick P.C. Lee, Tao Ouyang, Chao Shi, and Lilong Huang. Separating Data

via Block Invalidation Time Inference for Write Amplification Reduction in Log-Structured Storage. In *Proceedings of the USENIX Conference on File and Storage Technologies*, pages 429–444, 2022.

[45] Western Digital. Western Digital Ultrastar DC ZN540 Data Sheet. https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/collateral/data-sheet/data-sheet-ultrastar-dc-zn540.pdf, 2021.

[46] Wikipedia. MySQL. https://en.wikipedia.org/wiki/MySQL, 2023.

[47] Michael Wu and Willy Zwaenepoel. ENVy: A Non-Volatile, Main Memory Storage System. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, page 86–97, 1994.

[48] Gala Yadgar, Moshe Gabel, Shehbaz Jaffer, and Bianca Schroeder. SSD-Based Workload Characteristics and Their Performance Implications. *ACM Transactions on Storage*, 17(8):1–26, 2021.

[49] Jing Yang and Shuyi Pei. Thermo-GC: Reducing write amplification by tagging migrated pages during garbage collection. In *Proceedings of the IEEE International Conference on Networking, Architecture and Storage*, pages 1–8, 2019.

[50] Jing Yang, Shuyi Pei, and Qing Yang. WARCIP: Write amplification reduction by clustering I/O pages. In *Proceedings of the ACM International Conference on Systems and Storage*, pages 155–166, 2019.

[51] Jingpei Yang, Rajinikanth Pandurangan, Changho Choi, and Vijay Balakrishnan. AutoStream: Automatic stream management for multi-streamed SSDs. In *Proceedings of the ACM International Systems and Storage Conference*, pages 1–11, 2017.

[52] Ting Yao, Yiwen Zhang, Jiguang Wan, Qiu Cui, Liu Tang, Hong Jiang, Changsheng Xie, and Xubin He. MatrixKV: Reducing Write Stalls and Write Amplification in LSM-tree Based KV Stores with Matrix Container in NVM. In *Proceedings of the USENIX Annual Technical Conference*, pages 17–31, 2020.

A Artifact Appendix

Abstract

MiDAS is a migration-based data placement technique with adaptive group number and size configuration for reducing garbage collection overhead in various log-structured systems. Notably, MiDAS is currently implemented within the FTL of the real SSD prototype. For artifact evaluation, we provide our source code and the trace file. Please refer to the README file at https://github.com/dgist-datalab/MiDAS.

Scope

The artifact includes all the necessary source code required to run MiDAS as well as the FIO-based workload trace file used in this study. You can quickly test MiDAS using this trace.

Contents

We provide two Git repositories related to MiDAS: the SSD prototype-based implementation and the trace-driven simulation. If you evaluate MiDAS using the SSD prototype-based implementation, you can measure not only the WAFs but also I/O performance and the overhead associated with the CPU and memory for system execution. Meanwhile, using trace-driven simulations, you are limited to WAF evaluations only. Here, we will explain the contents based on the SSD prototype-based implementation. There are four main c files to run MiDAS:

- midas.c: adaptably changes group configuration and regularly checks irregular patterns at runtime (see §4.5).
- model.c: constructs UID, predicts WAF using MCAM, and runs GCS algorithm to find the best group configuration based on the observed workload patterns (see §4.3, §4.4 and §4.5).
- gc.c: selects a victim of GC based on the victim selection policy and moves live blocks into subsequent groups (see §4.1).
- hot.c: constructs a hot filter to separate hot blocks from cold blocks by monitoring incoming data blocks (see §4.2).

Hosting

We provide the public Git URLs, and the commit hashes for each repository used during artifact evaluation. MiDAS is implemented in both a real SSD prototype and trace-driven simulation. For quick testing of MiDAS, particularly for evaluating WAF, using the trace-driven simulation is sufficient. Additionally, we provide a public Zenodo URL for downloading the trace file of the FIO benchmark used in our evaluation.

- Emulated SSD prototype
  https://github.com/dgist-datalab/MiDAS
  `14be7bf7b01fad8db023622e4598fb9e30d8024f`
- Trace-driven simulation
  https://github.com/sungkyun123/MiDAS-Simulation
  `a22626529ad625eb4ddbab298752b9116fe6d05a`
- FIO-based workload trace file
  https://zenodo.org/record/10409599

Requirements

Hardware requirements. We use the Xilinx Virtex® UltraScale™ FPGA VCU108 platform and customized NAND flash modules. The customized NAND flash modules used in this paper are not publicly or commercially available. Therefore, you may need your own NAND modules compatible with VCU108 and adequate modifications to the hardware backend to replicate this work. Acknowledging the challenge for most researchers to replicate our experimental setup, we offer an alternative emulation of the prototype via a memory-based approach (RAM drive). DRAM must be larger than the device size of trace files + an extra 10% of the device size for the data structures and OP region to test the trace files. For example, you need 140GiB size of DRAM to run the trace file with a 128GiB device size.
Software requirements. There are little special software requirements to run MiDAS and you only need to install some packages using `apt`. The README file in the repository describes detailed instructions for the installation.