IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 18, 2021, accepted May 25, 2021, date of publication May 28, 2021, date of current version June 8, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3084851

Dynamic Hot Data Identification Using a Stack Distance Approximation

HYEONJI HA, DAEUN SHIM, HYEYIN LEE, AND DONGCHUL PARK, (Member, IEEE)
Department of Software, Sookmyung Women's University, Seoul 04310, South Korea

Corresponding author: Dongchul Park (dpark@sookmyung.ac.kr)

This work was supported in part by the National Research Foundation of Korea (NRF) Grant through the Korean Government [Ministry of Science and ICT (MSIT)] under Grant 2020R1F1A1048485, and in part by Sookmyung Women's University Research under Grant 1-1903-2021.

ABSTRACT Though various applications such as flash memory, cache, storage systems, and even indexing for enterprise big data search, adopt hot data identification schemes, relatively little research has been expended into holistically examining alternative strategies. Rather, researchers tend to classify hot data simplistically by considering one or more frequency metrics, thereby disregarding recency, which is also an important consideration. In practice, different workloads mandate different treatment to achieve effective hot data decisions. This paper proposes a *dynamic* hot data identification scheme that adopts a workload stack distance approximation. Stack distance is a good recency measure, but it traditionally requires high computational complexity as well as additional space. Since stack distance calculation efficiency is a core component for our dynamic feature design, this paper additionally proposes a stack distance approximation algorithm that significantly reduces both computation and space requirements. To our knowledge, the proposed scheme is the first *dynamic* hot data identification scheme which judiciously assigns more weight to either recency or frequency based on workload characteristics. Our experiments with diverse realistic workloads demonstrate that our stack distance approximation achieves excellent accuracy (up to a 0.1% error rate) and our dynamic scheme improves performance by as much as 49.8%.

INDEX TERMS Bloom filter, flash memory, hot data, hot data identification, SSD, stack distance.

I. INTRODUCTION

Hot data identification is a paramount issue in numerous fields [1]. For example, NAND (Not-AND) flash-based storage devices, such as SSDs (Solid State Drives) and USB (Universal Serial Bus) flash drives, must adopt an intermediate software layer named FTL (Flash Translation Layer) to hide NAND flash memory idiosyncrasies. The FTL performs logical-to-physical address mapping, executes a GC (Garbage Collection) algorithm to reclaim invalid (i.e., garbage) pages, performs WL (Wear Leveling) logic to improve flash memory's life span [11]. Since both GC and WL algorithms utilize a hot data identification mechanism, hot data identification algorithm effectiveness and efficiency have a critical impact on flash memory performance, as well as reliability (i.e., life span) [3], [4].

Effective data placement is an additional, excellent application for hot data identification. This applies to data placement mechanisms between DRAM (Dynamic Random Access Memory) and HDD (Hard Disk Drive) [5], [6], between SLC-MLC (Single Level Cell-Multi Level Cell), and flash-PCM (Phase Change Memory) hybrid SSD design [7], [8]. Intuitively, hot data should be placed on faster media. Analogously, cold data should be placed on slower media. Recently, big data applications have adopted hot data identification for fast (i.e., real-time or near real-time) enterprise searching through efficient indexing mechanisms. Elasticsearch, the most popular enterprise search engine worldwide, employs a hot-warm architecture that efficiently accommodates huge indices [9], [10]. With Elasticsearch, currently indexed indices with high search volumes are placed on hot nodes. Similarly, indices with relatively lower search volumes and no indexing are placed on warm nodes. NoSQL (Not only Structured Query Language) database such as Alibaba's ApsaraDB is another big data application adopting the hot and cold data separation mechanism to reduce storage costs [11], [12]. In addition to these applications, hot data identification has significant potential to be utilized in many other fields [13].

However, because most researchers only tend to consider reference frequency, hot data identification is suboptimal. Specifically, if any data have been accessed more than a

The associate editor coordinating the review of this manuscript and approving it for publication was Liang-Bi Chen.

predefined number of times (i.e., threshold), the logic simplistically classifies the data as hot data, otherwise, it defaults to cold data. This hot definition is simple but ambiguous. Assume two LBAs (Logical Block Address) have an equal access count within a given period. Further suppose: (1) a previously heavily-accessed LBA is never subsequently accessed and (2) another LBA is heavily accessed very recently. If a hot data identifier makes a simplistic hot data decision based on only frequency, the LBA accesses are indistinguishable. Hence, hot data identification scheme should consider both recency and frequency [1].

To resolve this problem, an MBF (Multiple Bloom Filter-based hot data identification scheme) was proposed to capture both recency and frequency [1]. BF (Bloom Filter) is a bit array to space-efficiently check if an element has previously appeared in a set at the cost of accuracy [13]. The MBF scheme adopted multiple BFs. MBF suggests considering recency equally with frequency for effective hot data identification. To capture both considerations, it proposed a new data structure (i.e., multiple bloom filters) and recorded information (i.e., LBA hash values) in the bloom filters, selecting one bloom filter in a round robin manner for each request. However, MBF does not effectively capture both recency and frequency. In particular, it loses considerable recency information due to its bloom filter selection mechanism. Specifically, MBF works as follows: For each request, it chooses one bloom filter from multiple ones *sequentially* (i.e., round robin fashion) and stores an LBA's hash value in it. After a predefined time interval(for instance, every 4,096 requests), it selects one bloom filter (i.e., named a reset bloom filter) and erases (i.e., resets) all information in it. It then resumes the recording process by sequentially selecting one bloom filter. This mechanism effectively captures frequency information, but not recency information, because the sequential bloom filter section inevitably destroys recency information [21]. The reset BF retains the most recent access LBA information since it was reset recently. Similarly, the right next BF (i.e., a next reset BF candidate) has stored all access LBA information for the longest period. However, except the reset BF, all other BFs retain diverse LBA information accessed from the most recent period (i.e., predefined time interval) to the oldest period according to the reset time of each BF because of the aforementioned sequential BF selection mechanism. This results in recency information destruction.

For the best recency capturing, MBF stays with one bloom filter during one predefined period (i.e., does not go to the next bloom filter for each request) and keeps recording all accessed LBA information in it. After one period, similarly it chooses the next bloom filter, and stores all referenced LBA information in it during the next period. This revised mechanism resolves MBF's recency capturing problem. However, it has a pitfall: it almost totally loses frequency information during that period (i.e., MBF with this mechanism captures no frequency information). Trading effective recency capture for frequency capture is an inborn MBF limitation. Thus, frequency information capture unfortunately compromises recency information capture, together with correct frequency capturing. This compromise is avoidable.

To resolve MBF's inevitable compromise, this paper proposes a dynamic hot data identification scheme that exploits a stack distance approximation. To our knowledge, the proposed scheme is the first *dynamic* hot data identification scheme that adapts to workloads. This scheme consists of three main components: a workload analyzer, a weight allocator, and a hot/cold identifier.

Like MBF, our proposed hot/cold data identifier also adopts multiple bloom filters and multiple hash functions. Multiple bloom filters enable capturing both recency and frequency. However, a principle difference between our dynamic scheme and MBF lies with a bloom filter selection mechanism. Our proposed scheme judiciously selects one of the bloom filters, assisted by our workload analyzer and weight allocator.

A workload analyzer provides a workload characteristics analysis by adopting a stack distance. Here, the stack distance is the number of unique objects accessed between two successive access to the same object. Recency is a more valuable factor for temporally localized access patterns and a stack distance is a good measure of this temporal locality. Thus, a small average stack distance across all references indicates a good temporal locality, which implies workloads with a small stack distance should consider recency a more important factor. We adopt a stack distance to instantiate adaptiveness. The stack distance can be used as a measure of recency, but requires both high computational complexity and high space consumption. Therefore, an efficient stack distance calculation is a key requirement for our dynamic feature design. Thus, this paper also proposes a stack distance approximation algorithm. It employs only one hash table that maintains simple information for each bucket and produces an approximated stack distance through very simple calculations. Consequently, the proposed approximation mechanism significantly reduces both overheads and exhibits significant performance with very high accuracy.

Based on the workload analysis, the weight allocator dynamically assigns more weight to either recency or frequency by intelligently (not merely sequentially) choosing each bloom filter. Moreover, unlike MBF, our preliminary work [1] which adopted a linearly-decreasing recency weight function, we choose an exponentially decreasing weight function to assign more realistic recency weight to each bloom filter. Consequently, this dynamic scheme more effectively captures recency as well as frequency.

The main contributions of this paper are as follows:

- ***A dynamic hot data identification scheme:*** Our proposed dynamic scheme effectively captures both recency and frequency together by judiciously selecting a bloom filter. When workloads have a small stack distance, it assigns more weight to recency by choosing a reset bloom filter storing the most recent data. Alternatively, when workloads exhibit larger stack distance, it assigns more weight to frequency by sequentially choosing the next bloom filter. Consequently,

the proposed scheme resolves MBF's inborn limitation (subsection III-A).

• ***A stack distance approximation algorithm:*** An efficient stack distance computation is a core concept in our dynamic design. For compactness, it employs one hash table that maintains simple information and performs very simple calculations (i.e., simple two value averages) to reduce computational complexity dramatically ($O(1)$). Weight assignment is fundamentally based on this approximation algorithm. Our extensive evaluation demonstrates it exhibits excellent performance with very high accuracy (subsection III-C).

• ***A novel baseline scheme:*** All existing baseline algorithms [1], [14] are static schemes. Thus, we propose a dynamic baseline scheme that employs a stack distance as a dynamic feature and an exponentially decreasing weight function to consider recency weight. Unlike existing approaches that require high computational complexity due to a batch decay mechanism, this baseline algorithm does not incorporate a batch decay (subsection III-E).

The remainder of this paper is organized as follows. Section II gives a bloom filter overview and two locality measures (i.e., stack distance and inter-reference distance). It also describes existing hot data identification schemes. Section III explains the design and operations of our proposed dynamic hot identification scheme and a novel baseline scheme. In addition, it describes our stack distance approximation algorithm, one of our core features. Section IV provides a variety of experimental results and analyses. Section V concludes the discussion.

II. BACKGROUND AND RELATED WORK

This section explains localities and discusses existing hot data identification schemes.

A. TWO LOCALITY MEASURES

A stack distance is the depth from which a certain reference must be extracted from a stack. It can be calculated from the number of unique accesses between two accesses to the same address in the trace. This has also been referred to as reuse distance. Given t_0, the stack distance definition, $sd(t) = |z|$, where z is the set of distinct references between t_0 and t:

$$z = \{ref(\tau) | t_0 < \tau < t\} \tag{1}$$

This stack distance concept was originally designed for virtual page modeling [15], but has been primarily used to model cache behavior because it presents temporal locality information.

Similarly, inter-reference distance (IRD) is the number of other references between two references to the same address in the trace. That is, when at time t, $ref(t) = x$, find t_0 which is the last previous x reference time,

$$t_0 = max\{\tau | 0 \leq \tau < t \wedge ref(\tau) = ref(t)\} \tag{2}$$

The inter-reference distance definition: $ird(t) = t - t_0$ Both stack distance and inter-reference distance look similar. However, while stack distance only counts distinct accesses, inter-reference distance counts all references between two identical address accesses.

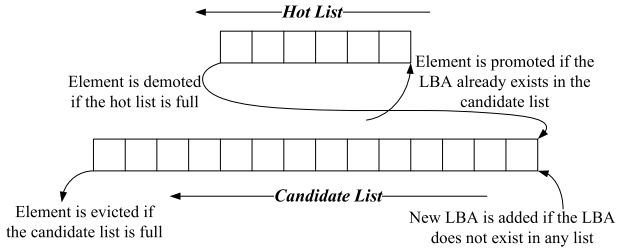


FIGURE 1. Two-level LRU list scheme. Performance heavily depends on the LRU size.

We use a stack distance to implement our proposed scheme's dynamic feature. However, since the stack distance involves very high complexity, we adopt the inter-reference distance to design our approximation algorithm.

B. BLOOM FILTERS

A bloom filter (BF) is a bit vector designed to quickly check if an element already exists in a set [13]. It is widely used to test so-called membership. Space efficiency is an important BF factor and it is achieved at the cost of accuracy. Therefore, a BF produces the following probabilistic results; the element either "(1) definitely does not exist" or "(2) maybe exists" in the set. When a given element is not in the set, a BF may provide a wrong positive answer, called a false positive (i.e., BF says the element exists in the set, when it does not). However, a basic BF *never* provides a false negative (i.e., when a BF indicates an element does not exist in the set, it does not).

A BF has three design parameters: a BF size (M), the number of hash function (K) and the number of distinct element (N). The BF is an array of M bits. All bits are initialized to zero. In addition, it needs K independent hash functions which produce K bit positions (i.e., K hash values) in the M bits array set to 1.

A BF operates as follows: first, it gets the K bit positions by feeding a given element to all K hash functions. Second, it goes to each bit position and checks the corresponding bits in the array of M bits. If any bit out of the K bit positions in the array is 0, this guarantees the element does not exist in the set because a BF never produces false negative (i.e., all the K bits would have been set to 1 if the element existed). On the contrary, if all K bits are 1, we must consider two possibilities: either (1) positive (i.e., the element exists in the set) or (2) false positive (which results from other element insertions). Fortunately, probability of a BF false positive is, in general, very low [16], [17]. Thus, the answer to the query is highly likely to be positive. We can achieve a very low false positive probability by adjusting the aforementioned three design parameters [1].

C. RELATED WORK

Flash Memory Server (FMS) is an out-of-place-update scheme to reduce the number of flash memory erase

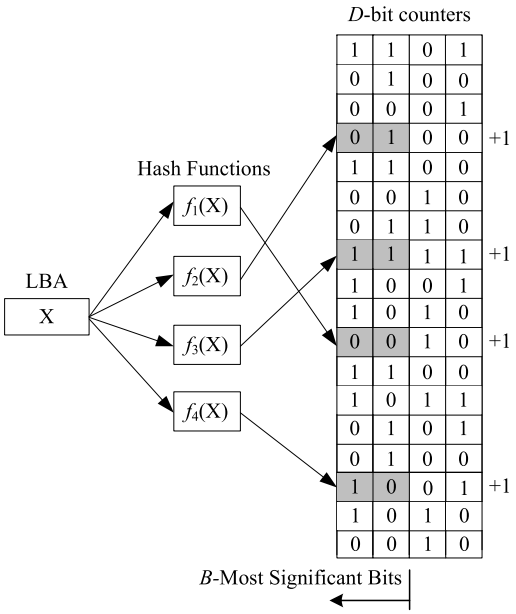


FIGURE 2. Multiple Hash Function scheme. This does not consider a recency factor. Here, D = 4, B = 2, and K = 4.

operations [18]. For efficient garbage collection, it classifies data into three categories: read-only, hot, and cold data. While FMS exhibits good data classification performance, it consumes significant memory space because it necessarily keeps the last access time information for all LBAs.

A two-level LRU (Least Recently Used) scheme resolves FMS' limitation [19]. It employs two lists, a hot list and a candidate list which are fixed size and operate under LRU policies. Though this two-level LRU scheme requires less memory than FMS, performance totally depends on both LRU list sizes. In addition, its LRU policy emulation involves high computational complexity. It is generally recognized that even a simple LRU (doubly-linked list) queue requires approximately 25% overhead to update the LRU queue on every record access [20].

A multiple hash function scheme (Figure 2) adopts *K*-hash functions and single BF with a *D*-bit counter. Each BF bit position corresponds to *D*-bit counter for capturing data access frequency. If any one bit of *B*-most significant bits is set to 1, this bit position is considered 1. If all *K*-hash bit positions are 1, the scheme classifies the data as hot data. Figure 2 presents an example adopting 4-bit counters and 2-most significant bits. Here, 4 will be its hot threshold value. If all four access count values of corresponding positions are greater than or equal to 4, the given LBA data are identified as hot. This scheme achieves a smaller memory footprint and entails lower computational complexity. However, it does not capture recency information.

A backward classification algorithm is an exponential smoothing-based hot and cold data identification algorithm for an in-memory database [20]. The main goal is to identify the *K*-hottest records efficiently among a large set of records and store them in main memory, keeping the remaining ones on secondary cold storage. It performs offline analysis to estimate record access frequency by scanning the log from present to past (i.e., backward). It also adopts a sampling approach to further reduce a processing overhead. The backward algorithm is an offline algorithm which may achieve higher accuracy. However, the offline algorithm can have limited applications and requires a significant storage space to store large log data temporarily for post-analysis. Consequently, this scheme specifically supports in-memory database systems [21].

Unlike existing work, our proposed hot data identification scheme is a dynamic algorithm that adapts to workloads. In addition, most existing schemes primarily consider a frequency factor, not recency. However, our proposed scheme considers both factors in order to classify hot data more effectively.

III. DYNAMIC HOT DATA IDENTIFICATION

This section describes our proposed dynamic hot data identification scheme

A. OVERALL FRAMEWORK

Figure 3 presents the overall architecture of our proposed dynamic hot data identification scheme. The proposed scheme consists of three major components: (1) a hot data identifier, (2) a workload analyzer, and (3) a weight allocator.

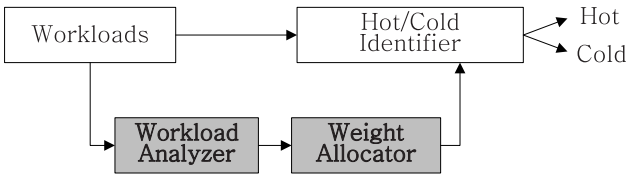


FIGURE 3. Overall architecture of our proposed dynamic scheme.

The hot data identifier is a main component and controls classifying given data as either hot data or cold data. To capture recency information as well as frequency information, it is composed of multiple independent bloom filters and multiple hash functions. Each bloom filter has different recency weights. Unlike MBF scheme [1] adopting a linearly decreasing weight assignment, each bloom filter is assigned an exponential smoothing-based recency weight to reflect a more practical situation.

The workload analyzer is a key component to implement our dynamic feature. For each incoming request, it analyzes workload characteristics by calculating a stack distance. The stack distance shows workload locality and can be used as a recency measure. Thus, for small stack distances, the workload analyzer assigns more weight to recency, otherwise more weight to frequency. Since calculating an exact stack distance involves very high complexity, we design a novel stack distance approximation algorithm that calculates the stack distance more efficiently. Consequently, it achieves lower computational complexity as well as space saving.

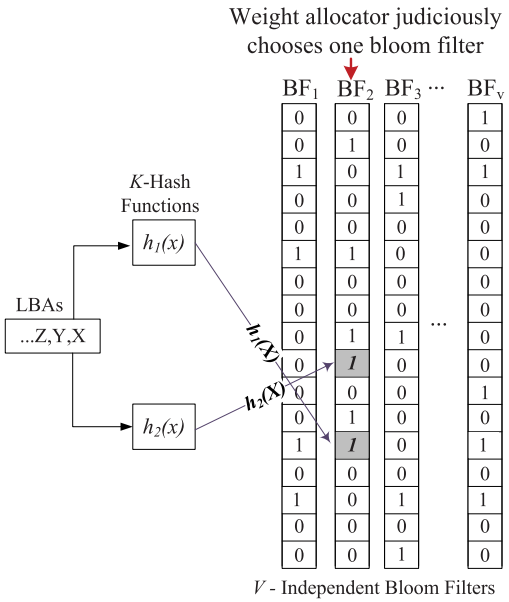


FIGURE 4. Our framework and its operations. It dynamically chooses one bloom filter for each request based on workload analysis. Here, K = 2.

Once this workload analyzer produces a recency weight value, the weight allocator assigns the corresponding weight to recency by probabilistically selecting each bloom filter in the hot data identifier. This is a key process to make a critical distinction between the proposed dynamic scheme and the existing MBF. Since MBF chooses each bloom filter sequentially, it inevitably loses recency information. However, our proposed scheme can carefully select one bloom filter with the help of the combined workload analyzer and the weight allocator.

Overall working processes are as follows: once any request arrives, the workload analyzer first performs a workload characteristic analysis by calculating a stack distance and generating a recency weight. Second, the weight allocator adopts the recency weight and probabilistically chooses one of the bloom filters. Third, the hot data identifier records the hash values to the bloom filter the weight allocator selects. Finally, if the combined (i.e., frequency and recency) hot data value is greater than or equal to the predefined hot threshold, the hot data identifier classifies the given data as hot data, otherwise cold data. The following subsections describe each component in more detail.

B. HOT DATA IDENTIFIER

Figure 4 illustrates our hot data identifier. It employs a set of V-independent bloom filters (BFs) and K-independent hash functions to capture both frequency and finer-grained recency. Each BF consists of M-bits to record K-hash values. The basic operations are as follows: whenever a request is issued, the K-hash functions hash the request's LBA. Each hash value ranges from 1 to M, respectively corresponding to an M-bit BF bit position. Thus, K-hash values set the corresponding K-bits to 1 in the BF the weight allocator selects

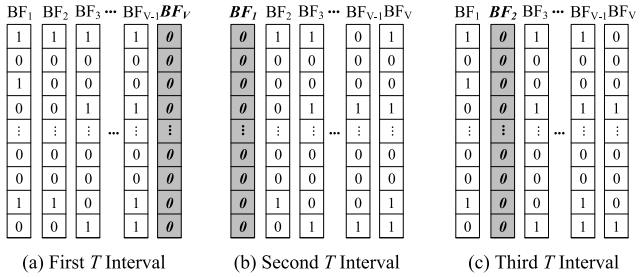


FIGURE 5. Our aging mechanism. Here, a gray bloom filter corresponds to a reset bloom filter.

(subsection III-D describes this weight allocator). In addition, it periodically selects one BF (i.e., the reset BF) in a round robin manner and erases all information in the selected BF to incorporate a decay effect.

1) FREQUENCY CAPTURING

The proposed hot data identifier does not maintain any counters for each LBA to record access counts. Instead, it simply performs a membership query in each of the V-independent BFs for each LBA. The number of BFs with a 'positive' result (i.e., the LBA has already appeared) out of V-BFs can express its frequency. Assuming the LBA hash values appear in r ($0 \leq r \leq V$) of the BFs out of V-BFs, we say the corresponding LBA has appeared r times before. This implies a maximum count value is directly associated with the number of BFs (i.e., V) and it is configurable (this BF number impact will be explored in our experiment (subsection IV)).

For increased frequency capture effectiveness, when choosing one of the V-BFs to store the K-LBA hash values, if the selected BF shows a 'positive' result, our method sequentially examines the next BFs until it finds one that has not recorded the LBA. This sequential examination minimizes recency analysis disruption. If it finds such an appropriate BF, it records the hash values to the BF. If it turns out that all (or a predefined number of) BFs have already contained the LBA information, our scheme simply classifies the data as hot (named a shortcut decision) and skips further processing including BF checking or recency weight assignment, since this definitely exceeds the threshold. This shortcut decision reduces computational overhead. For hot data identification, the information required in our scheme is simply whether the value r is larger than the threshold, not an accurate count in the case of $r \geq V$. Thus, when r is larger than a threshold, it simply passes the frequency check.

2) RECENCY CAPTURING

Since our hot data identifier does not maintain LBA counters, a different aging mechanism must be devised to capture recency information. Figure 5 illustrates our aging mechanism for decaying old information. First, consider that we adopt V-independent BFs and that LBA hash values are recorded to each BF in a round-robin manner during a predefined interval T. An interval T represents a fixed number of consecutive requests, not a time interval. As in figure 5 (a),

after an interval T, a BF that has not been selected for the longest interval is chosen and all bits in the BF (i.e., BF_V) are reset to 0. After the reset, the LBA hash values are again recorded to all BFs including the reset BF. After another interval T, the next BF (BF_1) is selected and all the bits are reset (figure 5 (b)). Similarly, after the next T interval, the next BF (BF_2) is chosen in a right cyclic shift manner and all information is erased (figure 5 (c)).

Figure 6 presents a recency coverage for each independent BF while BF_V is a reset BF. The reset BF (BF_V) records LBA information accessed during only the latest $1T$ interval. The previously reset BF (BF_{V-1}) can record the LBA information accessed during the last $2T$ intervals. Similarly, the BF_1 which will be chosen as a next reset BF immediately after this period can cover the longest interval $V \times T$. This means BF_1 records all LBA information for the last $V \times T$ intervals.

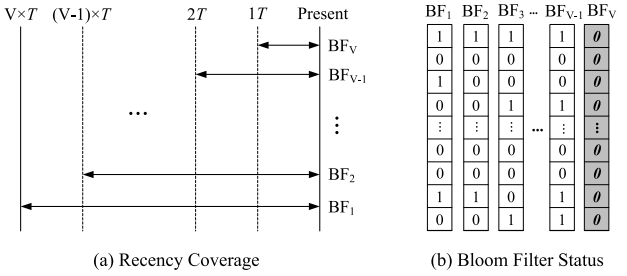


FIGURE 6. Recency coverage for each bloom filter. Here, a gray bloom filter (i.e., reset bloom filter) contains the most recent information.

Each BF has a different recency weight. A recency value is associated with a frequency value for hot data identification. The reset BF (BF_V) records most recent access information. Hence, a highest recency weight must be assigned to it, whereas a lowest recency weight is assigned to the BF that is selected as a next reset BF (BF_1) because it recorded LBA access information for the longest time period. We intend to use a recency value as a frequency value weight to produce a final value combining both. Consequently, even though two different LBAs have appeared once in BF_V and BF_1 respectively, both frequency values are regarded differently. Unlike MBF, this dynamic scheme adopts an exponential smoothing-based weight assignment mechanism. Subsection III-D describes this weight assignment mechanism in more detail.

3) A BLOOM FILTER SIZE

According to [17], given both hash function count (K) and the number of unique elements (N), we can theoretically estimate an optimal BF size (M) using the following formula,

$$M = \frac{KN}{\ln 2} \tag{3}$$

Obviously, a BF size (M) must be sufficiently larger than an element set size (N). For example, assume we adopt two hash functions ($K = 2$) and there are 4,096 unique elements ($N = 4,096$). Here, we should adopt an 11,819-bit vector as a correct BF size to minimize a false positive probability.

This basic BF only allows element addition, not removal. Thus, as the number of input elements grows, the data recorded in the BF continuously accumulates, increasing a higher false positive probability. To reduce this error rate, the traditional BF requires a large BF size as the formula suggests. However, we cannot simply incorporate this approach in our proposed scheme. Our scheme's BFs retain a distinguishing feature: BF information can be removed. Specifically, our proposed scheme periodically removes all information in one BF. This prevents continuous BF data accumulation. Therefore, we can adopt an even smaller BF than the aforementioned traditional BF.

In our proposed scheme, the BF size is closely related to a false identification rate. To meet user application requirements, our BF size is configurable and not fixed. Specifically, for higher accuracy, we can increase BF size, consuming more memory. On the other hand, if very high accuracy is not a critical application requirement, we can reduce BF size and save memory space. To address this issue, we conducted memory size impact experiments in our experiment section.

4) A DECAY PERIOD

The multihash function scheme [14] consists of 4,096 BF entries each of which is composed of a 4-bit counter. It adopts 5,117 write requests (N) as its decay period. This is based on an expectation that the number of hash table entry (M) can accommodate all the LBAs corresponding to cold data within every N (where, $N \leq M/(1-R)$) write requests (here, R is a workload hot ratio and the authors assumed R is 20%) [14]. To verify this assumption, as in figure 7, we measured the number of unique LBA for every 5,117 write request under several real traces such as Financial1, Distilled, MSR and RealSSD (these traces will be explained in our experiment section). Figure 7 clearly shows that the number of unique LBAs frequently exceeds their BF size (4,096) for each unit period (every 5,117 write requests) under all four traces. This necessarily causes hash collisions and results in higher false positive probabilities.

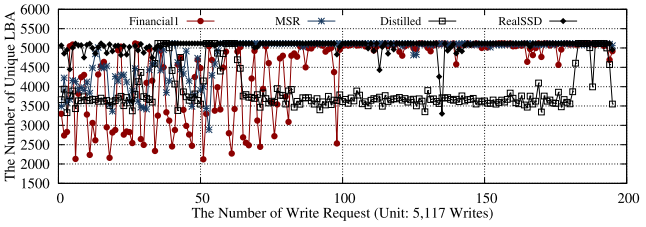


FIGURE 7. The number of unique LBAs within unit periods (5,117 writes) under various workloads. We observe a significant number of unique LBAs exceed the bloom filter size of 4,096.

We also measured the average number of unique LBAs within both 2,048 and 4,096 write request periods under the same workloads. They vary from 1,548 to 1,991 for 2,048 write request periods and from 3,014 to 3,900 for 4,096 write request periods. This is closely related with the average hot ratio of each workload: intuitively the higher

average hot ratio, the fewer unique LBAs. Based on these observations, we adopt M/V numbers of write requests for our decay interval T, where M is a BF size and V is the number of BF. To reduce false positive probabilities (i.e., error rates), the BF size must accommodate at least all unique LBAs that arrive in the last $V \times T$ interval. Thus, whenever M/V numbers of write requests are issued, one of the BFs are selected in a round robin fashion and all the bits in the BF are reset to 0. For example, assuming a BF size is 2,048, the decay period corresponds to 512 requests. Since memory consumption is also an important factor, we adopt a smaller (i.e., a half) BF size than the BF size for the multihash scheme.

C. WORKLOAD ANALYZER

A temporally localized workload pattern is a reference pattern where a more recently referenced block referenced in the near future. For this temporally localized pattern, recency is a more valuable factor and a stack distance is a good measure of this temporal locality [22]. Thus, a small average stack distance across all references indicates a good temporal locality, implying workloads with a small stack distance should consider recency a more important factor. Our proposed scheme utilizes this stack distance to implement our dynamic design. If the workload analyzer recognizes a small stack distance, it assigns more weight to a recency factor. Otherwise it assigns more weight to a frequency factor.

1) STACK DISTANCE APPROXIMATION

Our proposed stack distance approximation mechanism adopts a hash table where each hash bucket maintains the following three simple considerations: (1) a last index, (2) a last unique count, and (3) a reference.

The last index stores the index value at the time the corresponding reference is recently accessed. This last index is used to calculate inter-reference distance (IRD) efficiently. IRD is the number of other references between two references to the same address in the trace. Aside from this last index maintained in the hash table, we simply manage a global index. You can think of this global index as a time stamp which increments with each reference.

Similarly, the last unique count stores a quasi-unique reference count at the time the corresponding reference was recently accessed. It emulates the unique reference count between two identical references. To update this last unique count, we also keep another counter named a global unique count. We increase this count whenever a unique reference appears. Thus, this global unique counter also increments by 1 for each unique reference. Please note both the global index and the global unique counter are just simple counters maintained outside of the hash table.

The reference in the hash bucket stores accessed LBA information. The global unique counter utilizes this reference information in the hash table for efficiently checking if the reference is unique. That is, whenever a workload reference appears, we first check the corresponding hash bucket (i.e., the reference value in the bucket). If we find the same reference information in the bucket, which means the reference is not unique, the global unique count does not increase, otherwise the count increases by 1 because the reference is unique. Instead of this hash table, a bloom filter can be an alternative for quickly checking if the reference is unique.

Our proposed stack distance approximation operates as follows: when a request with a reference (i.e., LBA) is issued, we first check the corresponding hash bucket in the hash table and acquire the aforementioned three considerations. Second, if the hash bucket does not contain any existing information, we simply return 0 as the given LBA's stack distance because the LBA has not been previously accessed. Then, a global unique count increments since the given LBA is a unique LBA.

Third, if the hash bucket contains existing information, which means the given LBA has been accessed before, we calculate the IRD at a given time t, $IRD(t)$, by subtracting the last index from the current global index (i.e., $IRD(t) = a\ current\ global\ index - a\ last\ index$). Similarly, we approximate a unique reference count at time t, $aref(t)$, by subtracting the last unique count from a current global unique count (i.e., $aref(t) = a\ current\ global\ unique\ count - a\ last\ unique\ count$). Please note that a global unique count does not increase in this case since the given LBA is not a unique LBA. We next calculate an approximated stack distance of the given LBA at time t, $sd(t)$, by the following simple equation:

$$sd(t) = \frac{IRD(t) + aref(t)}{2} \tag{4}$$

Finally, we update both the last index and the last unique count in the corresponding hash bucket with the current global index and the current global unique count respectively. This approximation algorithm can significantly reduce the computational complexity to $O(1)$. More importantly, even though it is an approximated value, it is surprisingly accurate (Please refer to our experiment section IV).

However, during our extensive experiments, we found one limitation of our basic approximation algorithm: when it suddenly accesses an LBA which was assessed a very long time ago, it tends to produce a noticeably higher error rate (i.e., a higher stack distance than a precise stack distance).

Extensive investigation enabled us to determine that this problem occurs under a following specific workload access pattern: not only does it suddenly access an LBA that was assessed a very long time ago (causing an exceptionally large $IRD(t)$) but there are also many unique LBAs between the current access and the last access to the same LBA, where those unique LBAs were already repeatedly accessed before the last access (causing a relatively small unique reference count $aref(t)$). This is the worst case of our basic approximation algorithm.

Consequently, this scenario produces a higher approximation value than a precise stack distance. Precisely anticipating such an exceptional access workload pattern is very difficult. Based on our experiments and workload analyses with diverse

workloads, it was a rare access pattern, but this limitation requires compensation.

To smooth out this unexpected side effect, if the $IRD(t)$ is T (a predefined number, e.g., 5) times larger than the average $IRD(t)$, our revised approximation algorithm reduces the $IRD(t)$ value by half. In addition, if an unexpectedly large $IRD(t)$ continues appearing K (a predefined number, e.g., 50) times in a row, implying the workload access pattern keeps changing (not temporarily), it subsequently accepts the original $IRD(t)$ without smoothing.

With the help of this refined algorithm, our proposed approximation algorithm effectively deals with such an unexpected workload access pattern, improving accuracy.

2) COMPUTATIONAL COMPLEXITY ESTIMATION

Calculating a precise stack distance requires very high computational complexity ($O(NM)$, where N is the trace length and M is the number of distinct references in the trace) and consumes high space [15]. To resolve these limitations, Almasi *et al.* proposed a novel stack distance algorithm (not approximation) named hole-based algorithm [15]. It adopted an interval tree which is a balanced binary tree to calculate stack distance more efficiently. Although it notably reduced stack distance calculation complexity ($O(N \log(M))$, it is still so complicated that it is not useful to a hot data identification scheme which requires lower complexity [1], [13]. However, our proposed stack distance approximation mechanism significantly reduces computational complexity to $O(1)$.

D. WEIGHT ALLOCATOR

Once the workload analyzer produces a stack distance, the weight allocator first calculates recency weight at time t, λ, by the following equation:

$$\lambda = \left(\frac{1}{p}\right)^{\frac{sd(t)}{asd(t)}}, \quad 0 < \lambda \leq 1,\ p \geq 2. \tag{5}$$

Here, $asd(t)$ stands for an average stack distance at time t. We manage this average stack distance for each reference. After the recency weight is calculated, the weight allocator assigns this weight to recency by probabilistically picking one bloom filter (BF) in the hot data identifier to record hash values. This core process provides a critical distinction between the proposed dynamic scheme and MBF. Since MBF simply chooses each BF sequentially (i.e., round robin), it inevitably destroys recency information with each request. For optimal recency capture, MBF can choose one BF and keeps recording access LBA information (i.e.,hash values) in the selected BF during one predefined decay period. After the one period (e.g., 4,096 requests), MBF similarly chooses the next BF and executes all referenced LBA recording process during another decay period, etc. However, this mechanism results in complete LBA access frequency information loss during the corresponding decay period. This is the primary reason MBF has no alternative but to employ sequential BF selection mechanism for each request and not using one BF during one decay period, even though it partially loses recency information.

Our dynamic BF selection mechanism resolves this inborn limitation of MBF. The weight allocator operates as follows: for each request, once it produces recency weight (λ) with the equation 5, it either selects a reset BF containing most recent access information with a probability of $(\lambda \times 100)\%$, or another BF, but not the reset BF, with a probability of $((1-\lambda) \times 100)\%$. For more effective frequency capturing, when it chooses another BF except the reset BF, if the selected BF shows a 'positive' result (i.e., if the BF has already recorded the given LBA), it sequentially examines the next BFs until it finds an appropriate one that has not recorded the LBA. This sequential examination minimizes recency analysis disruption.

Therefore, our proposed scheme more intelligently selects a BF with the help of both the combined workload analyzer and the weight allocator.

E. A NEW BASELINE ALGORITHM

To our knowledge, our proposed scheme is the first dynamic hot data identification scheme. Thus, there are no baseline algorithms adopting a dynamic feature with workload adaptation. A Direct Address Method (DAM) [14] maintains counters for each LBA. Thus, it precisely captures the frequency information. However, with respect to recency, DAM does not have any mechanism to capture recency and moreover, it requires high space consumption. Window-based Direct Address Counting (WDAC) [1] can capture recency as well as frequency by adopting a sliding window. It assigns the highest recency weight to the present (i.e., head of the window). The weight linearly decreases toward the past (i.e., tail of the window). Although WDAC is the first baseline algorithm capturing both recency and frequency, it does not utilize a dynamic feature. Moreover, due to its batch decay mechanism, WDAC necessarily recalculates all combined values (named hot data index). This results in high computational complexity.

Unlike both DAM and WDAC, the proposed novel baseline algorithm adopts a dynamic feature. So, it adapts to workloads. It also employs a stack distance for dynamic design and an exponentially decreasing weight function, μ, as follows:

$$\mu = \left(\frac{1}{p}\right)^{\frac{sd(t)}{asd(t)} \times \tau}, \quad 0 < \mu \leq 1,\ p \geq 2. \tag{6}$$

This weight function is identical to that of our weight allocator (Equation 5) except that this adopts a control parameter, τ. Intuitively, the meaning of τ in this equation is that a hot data value is reduced to $1/p$ of the original value after every $1/\tau$ time steps. This control parameter, τ, allows a fundamental trade-off between recency and frequency effect. As τ approaches 0, this weight function moves toward a frequency-based policy. Alternatively, as τ approaches 1, it moves towards a recency-based policy. Specifically, when τ is 0, it ignores recency (i.e., only considers frequency). When τ is equal to 1, it aggressively considers recency, so recency

TABLE 1. System parameters and values.

Parameters	Dynamic	MBF	MHF	Baseline	DAM
BF Size	2^{11}	2^{11}	2^{12}	N/A	N/A
Number of BF	4	4	1	N/A	N/A
Decay Interval	2^{9}	2^{9}	2^{12}	N/A	2^{12}
Number of Hash	2	2	2	2	N/A
Hot Threshold	4	4	4	4	4
Recency Weight	$(1/2)^{x}$	2^{-1}	N/A	$(1/2)^{y}$	N/A

weight significantly drops from a present metric to a past metric.

Our dynamic baseline algorithm adopts a hash table to efficiently maintain a hot data value for each LBA. For each accessed LBA, if the given LBA is a newly accessed LBA (i.e., has not been accessed before), we assign an initial hot data value, c, to the corresponding hash bucket. If the given LBA hits the hash bucket (i.e., it has been previously accessed, so that the corresponding hash bucket already retains an existing hot data value), it first applies the exponentially decreasing recency weight, μ, to the existing hot data value by multiplying the existing hot data value by μ. This reflects a recency decaying effect to the existing hot data value as time progresses. It then adds an initial hot data value, c, to the decayed hot data value. If this newly calculated LBA hot data value is greater than or equal to a predefined hot data threshold, it classifies the LBA as hot data, otherwise, it classifies the LBA as cold data.

IV. EXPERIMENTS

This section provides diverse experimental results and comparative analyses.

A. EVALUATION SETUP

To conduct an extensive and objective evaluation, our proposed dynamic hot data identification scheme is compared with four other schemes: Multiple Bloom Filter-based scheme (hereafter, referred to as MBF) [1], Multiple Hash Function scheme (referred to as MHF) [14], our proposed dynamic baseline scheme (referred to as Baseline), and Direct Address Method (referred to as DAM) [14].

We partially consider MBF because it is somewhat redundant. That is, MBF and MHF were already evaluated thoroughly in our preliminary work [1] and demonstrated that MBF outperformed MHF in multiple respects(please refer to [1]). Therefore, we particularly focus on MBF because not only does the MBF base our dynamic scheme, but also it is now considered to be most well-known scheme. Moreover, our dynamic scheme resolves MBF's inborn limitations.

Two different baseline schemes are employed: baseline (our proposed) and DAM. Since our proposed scheme is the first dynamic hot data identification scheme, we primarily focus on our proposed dynamic baseline scheme. However, for a more objective evaluation, DAM is also evaluated.

Table 1 shows system parameters and their values. Both our proposed dynamic scheme (hereafter, refer to as Dynamic) and MBF adopt a bloom filter half the size of the MHF because they exhibit a better performance than MHF, even though it is smaller. However, we also evaluate these three schemes (i.e., Dynamic, MBF, MHF) with the same bloom filter size for a clearer understanding.

TABLE 2. Workload characteristics.

Workloads	Total Requests	Request Ratio (Read:Write)	Average Inter-Arrival Time
Financial1	5,334,987	R:1,235,633(22%) W:4,099,354(78%)	8.19 ms
MSRprxy	1,048,577	R:47,380(5%) W:1,001,197(95%)	N/A
Distilled	3,142,935	R:1,633,429(52%) W:1,509,506(48%)	32 ms
RealSSD	2,138,396	R:1,083,495(51%) W:1,054,901(49%)	492.25 ms
MSRprn	1,048,576	R:904,519(86%) W:144,057(14%)	N/A
MSRproj	1,048,576	R:797,152(76%) W:251,424(24%)	N/A
Financial2	3,699,195	R:3,046,113(82%) W:653,082(18%)	17.42 ms

For fair evaluation, the same number of requests (4,096) is assigned for a decay interval in DAM and MHF, while 512 requests are adopted for our decay period and an MBF decay period. Two identical hash functions are employed for our dynamic scheme, MBF, MHF, and baseline. Lastly, we assign 0.5 weight difference for each bloom filter in MBF. We also assign an exponentially decreasing weight for each bloom filter in our dynamic scheme and for each request in our proposed dynamic baseline.

For more objective evaluation, diverse (i.e., seven) real workloads (Table 2) are employed. Financial1 and Financial2 are from the University of Massachusetts at Amherst Storage Repository [23]. Financial1 is a write-intensive trace file. Financial2 is a read-intensive trace file. These trace files were collected from a financial institution's running online transaction processing (OLTP) application.

The Distilled trace file [19] shows general and personal usage patterns in a laptop such as web surfing, watching movies, playing games, and documentation work. This is from the flash memory research group repository at National Taiwan University. Since the MHF scheme only uses this trace file, we also adopt this trace for fair evaluation.

We also utilize MSR trace files made up of 1-week block I/O traces of enterprise servers at Microsoft Research Cambridge Lab [24]. We select *prxy volume 0* trace for a write-intensive workload and both *prn volume 0* and *proj volume 0* for read-intensive workloads [25].

Lastly, we employ a real solid state drive (SSD) trace file that consists of 1-month block I/O traces (hereafter, refer to as RealSSD trace file) of a desktop computer in the Center for Research in Intelligent Storage (CRIS) lab at the University of Minnesota–Twin Cities [1]. We installed DiskMon for Windows [26] to the computer and collected personal traces such as computer programming, running simulations, documentation work, web surfing, and watching movies.

The total requests in Table 2 correspond to the total number of read and write requests in each trace. These I/O requests also can be subdivided into several or more sub-requests (i.e., block requests). We adopt this block-level request for our experiments.

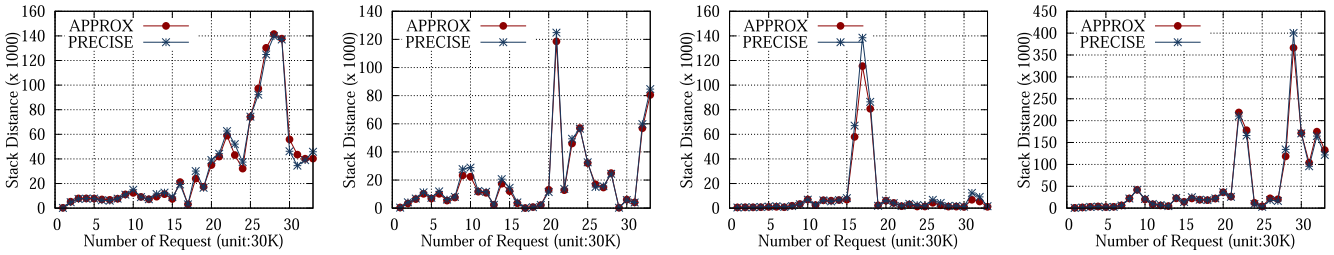


FIGURE 8. Average stack distance of both our proposed approximation mechanism and an existing precise mechanism. Here, APPROX stands for our proposed stack distance approximation.

B. PERFORMANCE METRICS

We measure an average stack distance to evaluate our proposed stack distance approximation performance by comparing it with a precise average stack distance. A hot ratio is the ratio of hot data to all data. This hot ratio metric is chosen to compare the performance of those schemes. However, an identical hot ratio of two schemes does not necessarily mean identical performance since hot data classification results from both schemes may differ. In other words, an identical hot ratio simply signifies the same amount of hot data compared to all data. It does not necessarily mean all classification results are also identical [1]. Thus, compensating for this limitation requires another evaluation metric, a false identification rate. For each request, we compare the identification result of each scheme. This enables us to perform a more precise analysis. Finally, runtime overhead also must be considered. To evaluate it, we measure average CPU usage.

C. RESULTS AND ANALYSIS

We discuss our evaluation results from multiple perspectives.

1) STACK DISTANCE APPROXIMATION

The stack distance approximation is one of our dynamic scheme's core algorithms. So, its performance is first evaluated with various real workloads. Figure 8 presents an average stack distance of both our approximation and a precise stack distance value. We measure this average stack distance with 1 million requests for each trace and plot it for each 30,000 request unit, which provides more thorough comparison.

As in figure 8, our approximation algorithm shows highly accurate, phenomenal performance. Our approximation values are very close to the precise stack distance for all four traces. We additionally evaluate the performance with three other traces and they all also exhibit a notable performance. So we omit them.

For a more extensive evaluation, we measure an average error with an average stack distance in parentheses for each trace as follows: 19 (35,831) for Financial1, 20 (19,744) for MSRprn, 27 (10,333) for MSRproj, and 15 (56,361) for Distilled trace. When we consider the average error together with the average stack distance, we can evaluate the performance of our approximation scheme more objectively.

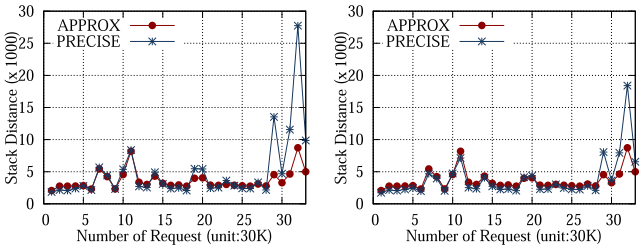


FIGURE 9. The effectiveness of our stack distance approximation smoothing algorithm under MSRprxy trace.

Moreover, considering the complexity of our approximation ($O(1)$) and that of the exiting state-of-the-art algorithm ($O(nlog(m))$ producing a precise value [15], even though ours is an approximation, this evaluation clearly demonstrates that our proposed approximation is applicable to many other fields, provided they do not require 100% accurate stack distance values.

Although our basic approximation algorithm shows remarkable performance under all traces, one trace exhibits a significantly higher error rate within a specific access range under MSRprxy trace. Specifically, figure 9 (a) depicts a range (i.e., between 29K and 33K) with a high error rate. Interestingly, except this specific range, our basic approximation algorithm normally exhibits outstanding performance under that trace. Extensive workload analysis investigation enabled us to identify the main reason for this problem; A specific workload access pattern that not only suddenly accessed an LBA it accessed a very long time ago (causing a very large IRD) but also experienced many unique LBAs between two accesses to the same LBA that were repeatedly accessed in the past (causing a relatively small unique reference count). This is the worst case of our basic approximation algorithm.

Figure 9 displays the approximation smoothing algorithm effectiveness. Referring to figure 9, the average error rate between the 29K and 33K range notably (43.7%) decreases. There may be a better solution further lowering the error rate, particularly under MSRprxy trace. However, such a specific solution can cause overfitting of our approximation algorithm, which decreases overall performance under the other general workloads. This suggests a general solution that uses configurable parameters in the approximation algorithm.

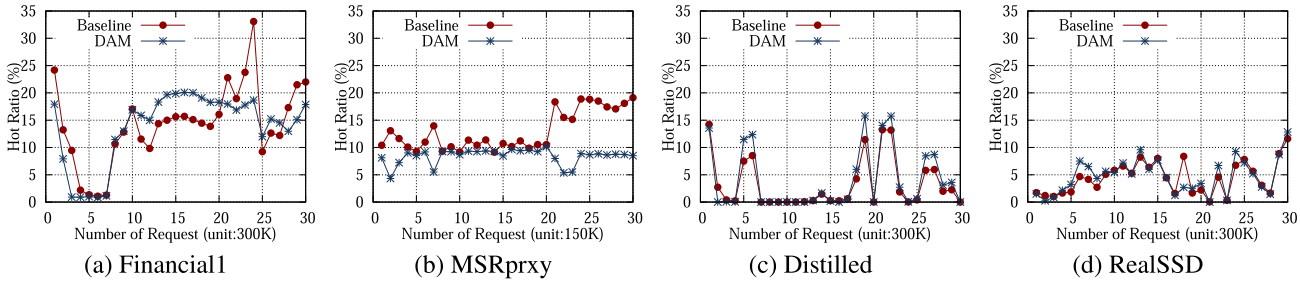


FIGURE 10. Hot ratios of two baseline algorithms.

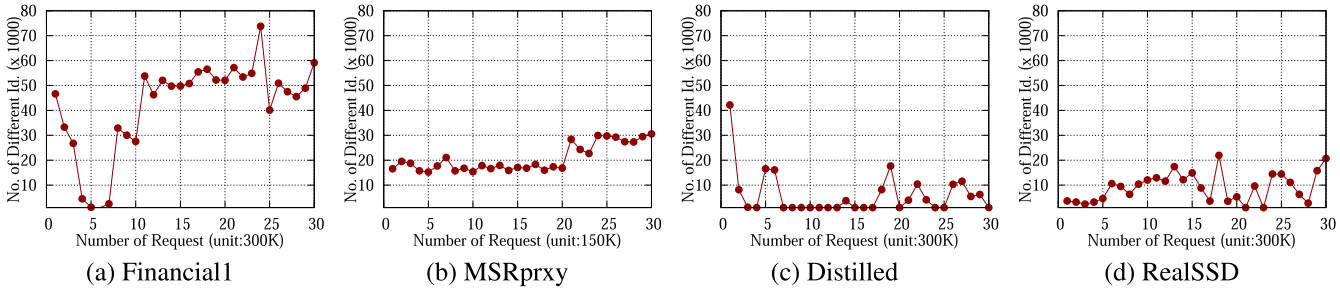


FIGURE 11. Number of different identification of two baseline algorithms.

2) BASELINE ALGORITHM

This subsection describes evaluation of two baseline schemes: the proposed baseline (refer to as baseline) and DAM. Figure 10 shows that the hot ratios of both schemes exhibit notably different results under Financil1 and MSRprxy (figure 10 (a) and (b)). On the other hand, they display very similar hot ratio patterns under Distilled and RealSSD traces (figure 10 (c) and (d)). However, identical hot ratios do not necessarily mean identical classification of both schemes. Thus, we need another experiment for a more comparative analysis: we count the number of different hot classification results during a specific unit time (150K or 300K requests). Zero means all identification results are identical. Please note that since MSRprxy traces in Figure 10, 11, and 12 do not have enough write requests, 150K unit time was employed instead of 300K unit time.

Figure 11 shows the number of different identification results between our baseline and DAM. Both figure 11 (c) and (d) demonstrate that performance can differ even though two hot ratios are very similar as in figure 10 (c) and (d).

3) PROPOSED SCHEME (Dynamic) VS. MHF

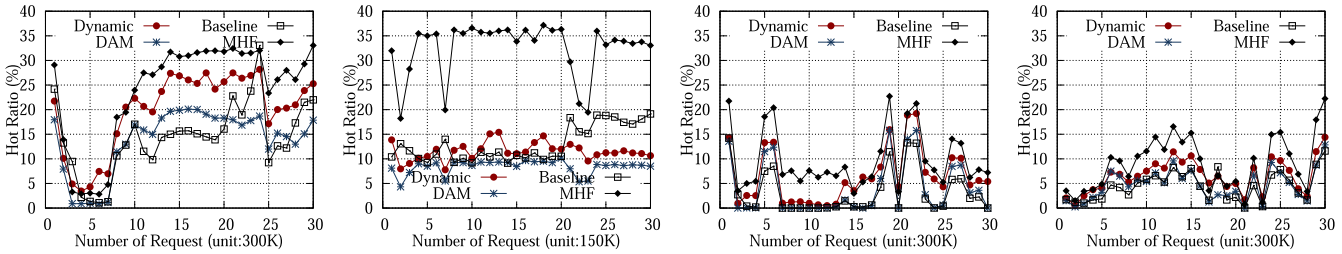
We evaluate four schemes in this subsection: Dynamic, MHF, Baseline, and DAM, particularly focusing on the proposed scheme (Dynamic) and MHF. We compare both schemes with Baseline as well as DAM to demonstrate our proposed baseline scheme is not an unfairly customized baseline scheme that is a best-fit for our scheme.

As in figure 12, our proposed scheme shows similar results to our baseline and even to DAM, while MHF tends to present notably higher hot ratios than Dynamic, as well as the two baselines under all traces. This results from a relatively higher false identification ratio of MHF. Due to a limited BF size, even though cold data requests increase, hash collisions can cause a higher chance of incorrect counter increments. Consequently, this results in higher hot ratios. As the MHF authors mentioned in their paper, the MHF scheme does not achieve good performance, especially when the hot data ratio in traces is low [14]. To bypass this limitation, they provided two suggestions: an increment of a hash table size or a shorter decay period. However, both solutions cannot become fundamental solutions because, not only does a larger hash table consume a more memory space, but also decreasing the decay interval introduces significant computational overhead [1]. Our scheme, on the other hand, resolves these limitations by adopting smaller, multiple, independent BFs.

Figure 13 presents false identification rates of both Dynamic and MHF by comparing both schemes with our proposed baseline. They exhibits the proposed scheme (i.e., Dynamic) shows even lower false identification rates than MHF by an average of 36.3%, 50%, 28.8%, and 38.6% under the four respective traces.

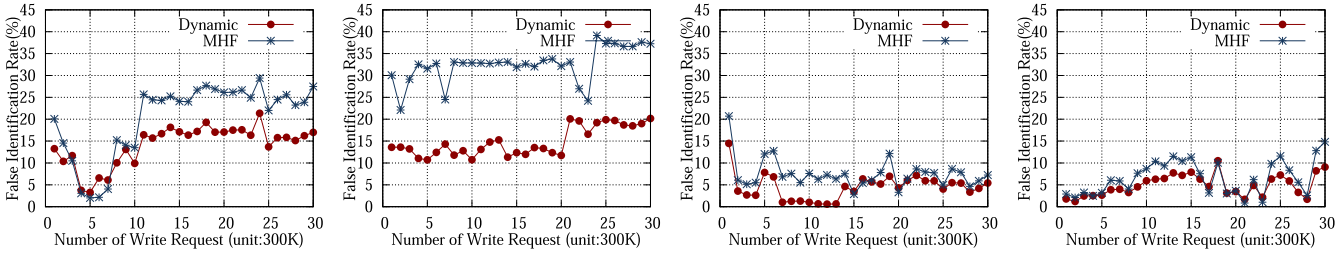
4) PROPOSED SCHEME (Dynamic) VS. MBF

Our dynamic hot data identification scheme is primarily based on MBF to resolve MBF's inborn limitation. Since extensive performance comparison of MBF with other existing schemes including MHF, WDAC, and DAM were already discussed in detail in [1], this subsection particularly focuses on MBF scheme (vs. our proposed scheme). Moreover, both MBF and MHF employed write-intensive workloads because



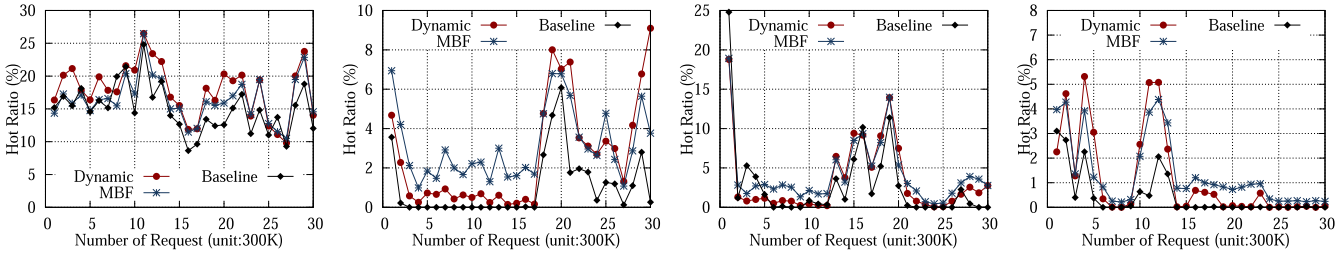
(a) Financial1 (b) MSRprxy (c) Distilled (d) RealSSD

FIGURE 12. Hot ratios of four schemes under various traces.



(a) Financial1 (b) MSRprxy (c) Distilled (d) RealSSD

FIGURE 13. False identification rates of both Dynamic and MHF.



(a) Financial2 (b) MSRprn (c) MSRproj (d) Distilled

FIGURE 14. Hot ratios of three schemes under various traces.

their design mainly considered NAND flash memory storage devices. On the other hand, our proposed scheme is not specifically designed for them and we adopt different (read-intensive) traces this time for more extensive comparison. As in table 2, Financial2, MSRprn, and MSRproj are newly adopted read-intensive workloads. For a more objective and fair comparison, we still employ one of the previous traces (Distilled) because only Distilled trace contains enough read requests (52%). Please note RealSSD trace also contains enough read requests (51%), but both Distilled and RealSSD traces show very similar workload characteristics because they were collected from personal computers for personal traces. Moreover, Distilled trace were collected from a different research group (i.e., National Taiwan University), whereas RealSSD trace were collected from one of our research group members (University of Minnesota–Twin Cities). Therefore, we choose Distilled instead of RealSSD for a more objective evaluation.

Figure 14 exhibits hot ratios of the proposed scheme (Dynamic) and MBF. For reference, our proposed baseline is included. Figure 14 (b), (c), and (d) clearly show Dynamic's hot ratios are similar to the baseline and corresponding false identification rates are undoubtedly lower than those of MBF (figure 15 (b), (c), and (d)). Financial2 trace shows an interesting result. Contrary to our expectation, MBF's hot ratio is closer to the baseline and a total average hot ratio of MBF (14.5%) is also closer to that of the baseline (13.6%) than that of our proposed scheme (15.2%). However, as in figure 15 (a), the false identification rate of Dynamic is lower than that of MBF by an average of 2%. This clearly demonstrates that an identical hot ratio does not necessarily mean an identical performance.

MBF can capture recency as well as frequency by recording LBA information to multiple bloom filters in a round robin fashion. This sequential bloom filter selection partially destroys recency information. As a result, MBF tends to perform better when the frequency is the more important workload factor. This is because MBF's sequential bloom filter selection policy captures frequency more effectively than recency. On the other hand, the proposed scheme dynamically

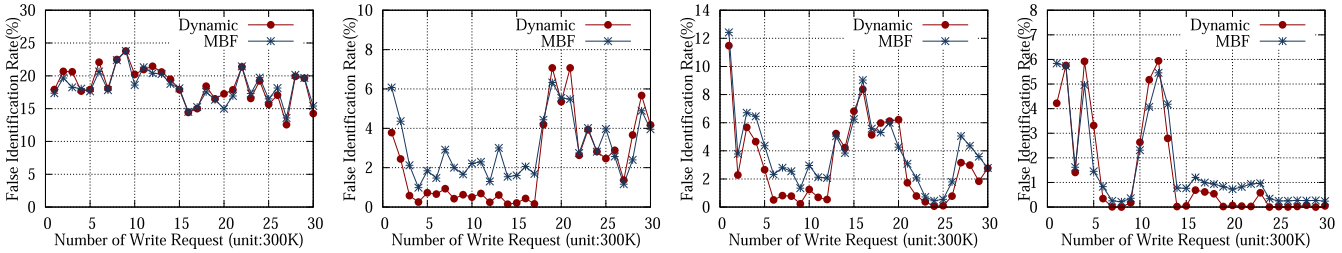


FIGURE 15. False identification rates of both Dynamic and MBF.

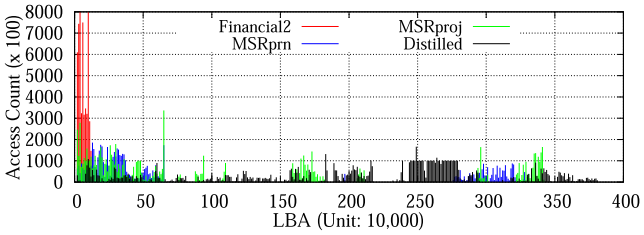


FIGURE 16. LBA access pattern for each trace.

chooses one of the bloom filters to effectively capture recency while still capturing frequency well, like MBF. Figure 16 displays workload access patterns (i.e., LBA access count) of four traces. Financial2 shows a very different access pattern from the other three: it heavily accesses a very limited LBA range (i.e., red histogram), while others access a very wide LBA range. The frequency factor is more important under Financial2 trace and consequently, MBF exhibits a comparable performance with the proposed scheme (figure 15 (a)). However, Dynamic clearly shows a better performance than MBF ((figure 15 (b), (c), and (d)).

5) BLOOM FILTER SIZE IMPACT

A bloom filter size is closely related to memory space consumption. This subsection investigates impact of the bloom filter size. The proposed scheme comprises 4 bloom filters each of which requires 0.25 KB (i.e., 2K×1-bit). That is, 4 bloom filters consume 1KB memory space. We increased each bloom filter size of the proposed scheme from 2K (i.e., 2,048 bit) to 8K (8,192 bits), and 4 bloom filters remain.

Figure 17 illustrates false identification rates of our scheme with different bloom filter sizes. As a bloom filter size increases, performance also improves. That is, our scheme with a larger bloom filter size exhibits a lower false identification rate. For instance, under Financial2 trace, the average error rates of 8K, 4K, and 2K bloom filter size correspond to 11.1%, 12.2%, and 16.1% respectively. This is very intuitive because a large bloom filter lowers hash collision probabilities, resulting in a lower false positive rate.

The BF size is not a fixed parameter in our scheme and is configurable to meet applications' requirements. For example, assuming an application requires the higher accuracy of hot data identification (i.e., lower false identification rate) as a critical factor, we can increase the bloom filter size although this consumes more memory. Conversely, if decent accuracy is acceptable, we can save memory space by adopting a smaller bloom filter size.

6) BLOOM FILTER NUMBER IMPACT

Both multiple bloom filters (i.e., data structure) and a bloom filter selection mechanism (i.e., algorithm) lie at the core of our proposed dynamic hot data identification scheme. Multiple bloom filters are responsible for capturing recency information and the number of bloom filters is associated with recency granularity. To explore its impact, we varied the numbers of bloom filters. For a more objective comparison, we fixed a memory space (1KB) and configured both bloom filter sizes and decay periods accordingly.

As in figure 18, as the number of bloom filters increases, false identification rates also increase. Since we fix a memory space, smaller size bloom filters are assigned to each scheme with more bloom filters. As a result, this causes higher error rates. Even though the scheme with more bloom filters can capture fine-grained recency, it offsets this advantage with higher error rates. However, if we do not fix the memory space in this evaluation, doubling the number of bloom filters from 4 to 8 for instance, while keeping the identical bloom filter size, our proposed scheme can benefit from the finer-grained recency [1].

7) COMPUTATIONAL OVERHEAD

Computational overhead is another important factor to evaluate for a hot data identification scheme. We measure average CPU usage to evaluate the computational overhead. We run three schemes (i.e., MBF, our proposed scheme, and baseline) on MacBook Pro (Intel Core i5 quad core@1.4GHz, 16GB RAM, Mac OS V.11.0.1) for each trace. For fair evaluation, whenever we conduct each experiment, we reboot our computer and stay for 30 minutes without running any applications. Then, we run each hot data identification scheme and measure average CPU usage with '*top*' Linux command.

Figure 19 exhibits average CPU usage for three schemes. Since all other traces show very similar performance patterns, we omit other traces except financial 1 and financial 2. As in figure 19, our proposed scheme (i.e., Dynamic) consumes more CPU than MBF by an average of 6.3% because it requires additional operations such as workload analysis and weight allocation to implement its dynamic feature.

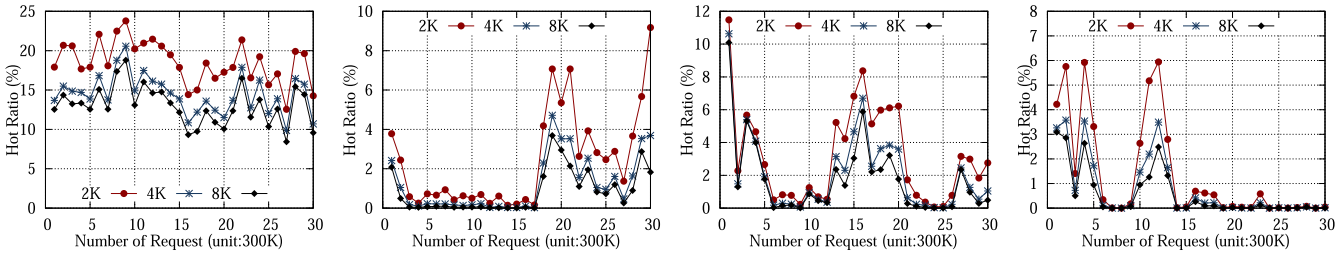


FIGURE 17. False identification rates of Dynamic with various BF sizes.

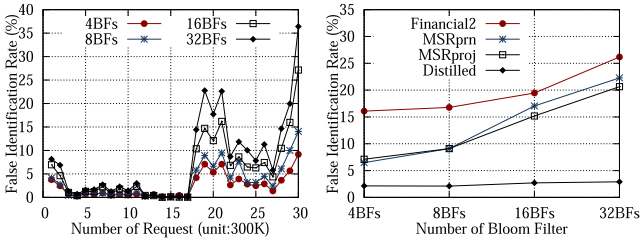


FIGURE 18. Performance change over various numbers of a bloom filter under various workloads.

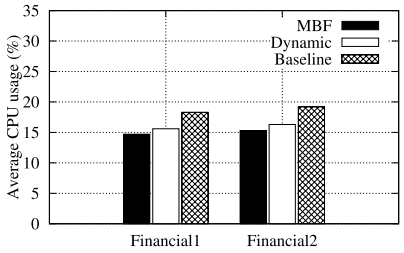


FIGURE 19. Average CPU usage for each scheme.

Our baseline algorithm consumes 17.5% more CPU than the proposed scheme. This mainly results from its precise stack distance calculation. On the other hand, the proposed scheme adopts our stack distance approximation mechanism which requires notably lower computational complexity.

V. CONCLUSION

This paper proposed a dynamic hot data identification that adapts to workloads. The proposed scheme adopts multiple bloom filters to effectively capture both recency and frequency. Each independent bloom filter is assigned a different (i.e., exponentially decreasing) recency weight and, as a result, each bloom filter has different recency coverage.

Unlike other existing schemes, the proposed scheme is a *dynamic* scheme and, to our knowledge, it is the first dynamic hot data identification scheme. To implement this core adaptive feature, we adopted a stack distance metric which is a measure of temporal localities. However, since calculating a precise stack distance requires very high computational complexity, we also proposed a stack distance approximation algorithm. This approximation algorithm not only notably reduced computational complexity, it also exhibited excellent accuracy. Therefore, we employed our stack distance approximation algorithm to analyze workload characteristics efficiently.

Once the stack distance is computed, our proposed scheme calculates a recency weight value based on the stack distance. Then, it dynamically (not sequentially) chooses one of the bloom filters by using the recency weight. For each request, the proposed scheme judiciously chooses the bloom filter and records the access LBA information in it on a workload basis. Consequently, the proposed scheme resolves MBF's inborn limitations.

This paper also suggested a more reasonable baseline algorithm. Unlike the existing baseline algorithms, it adapts to workloads by adopting an exponentially decreasing weighing function based on a stack distance. In addition, to reduce computation, it removes a batch decay operation existing schemes employ. Instead, it updates a hot data value for each LBA by applying an exponentially decreasing recency weight only to a corresponding LBA counter (i.e., hash bucket).

We conducted experiments in many respects using various real traces. Our proposed stack distance approximation exhibits excellent performance with high accuracy; especially 3 traces out of 7 traces exhibit error rates of less than 1% (0.7%, 0.1%, and 0.7% respectively). The proposed dynamic hot data identification scheme outperformed existing schemes by as much as 49.8%.

For our future work, we plan to apply our dynamic scheme to real-world applications including a new cache design and SMR (Shingled Magnetic Recording) drives that is next generation hard disk drives to get over HDD's capacity limitation. In addition, a machine learning-based hot data identification scheme design is another future work.

ACKNOWLEDGMENT

The authors would like to thank David Schwaderer (QuantumSafe, USA) for his valuable comments and proofreading.

REFERENCES

[1] D. Park and D. H. C. Du, "Hot data identification for flash-based storage systems using multiple Bloom filters," in *Proc. IEEE 27th Symp. Mass Storage Syst. Technol. (MSST)*, Denver, CO, USA, May 2011, pp. 1–11.

[2] D. Lee, J. Kwak, G. Lee, M. Jang, J. Jeong, K. Wang, J. Choi, and Y. H. Song, "Improving write performance through reliable asynchronous operation in physically-addressable SSD," *IEEE Access*, vol. 8, pp. 195528–195540, 2020.

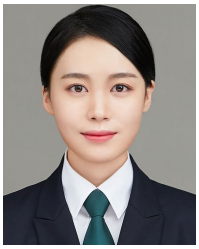
[3] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX ATC*, Boston, MA, USA, 2008, pp. 57–70.
[4] L.-P. Chang and T.-W. Kuo, "Efficient management for large-scale flash-memory storage systems with resource conservation," *ACM Trans. Storage*, vol. 1, no. 4, pp. 381–418, Nov. 2005.
[5] T. Kgil, D. Roberts, and T. Mudge, "Improving NAND flash based disk caches," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2008, pp. 327–338.
[6] J. Mogul, E. Argollo, and M. S. P. Faraboschi, "Operating system support for NVM+DRAM hybrid main memory," in *Proc. USENIX HotOS*, 2009, pp. 1–5.
[7] L.-P. Chang, "Hybrid solid-state disks: Combining heterogeneous NAND flash in large SSDs," in *Proc. Asia South Pacific Design Autom. Conf.*, Seoul, South Korea, Jan. 2008, pp. 428–433.
[8] G. Sun, Y. Joo, Y. Chen, Y. Xie, Y. Chen, and H. Li, "A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement," in *Emerging Memory Technologies*. New York, NY, USA: Springer, 2014, pp. 51–77.
[9] S. Bennacer. (2020). *Hot-Warm Architecture in Elasticsearch*. [Online]. Available: https://www.elastic.co/blog/hot-warm-architecture-in-elasticsearch-5-x
[10] J. Landis. (2019). *Implementing a Hot-Warm-Cold Architecture With Index Lifecycle Management*. [Online]. Available: https://ptran32.github.io/2020-08-08-hot-warm-cold-elasticsearch/
[11] Alibaba. (2021). *ApsaraDB for HBase*. [Online]. Available: https://www.alibabacloud.com/product/hbase
[12] A. A. Zeeshan. (2019). *Hot Data vs. Cold Data: Why It Matters?* [Online]. Available: https://www.alibabacloud.com/blog/hot-data-vs–cold-data-why-it-matters
[13] D. Park, B. Debnath, Y. Nam, D. H. C. Du, Y. Kim, and Y. Kim, "HotDataTrap: A sampling-based hot data identification scheme for flash memory," in *Proc. ACM SAC*, Trento, Italy, 2012, pp. 1610–1617.
[14] J.-W. Hsieh, T.-W. Kuo, and L.-P. Chang, "Efficient identification of hot data for flash memory storage systems," *ACM Trans. Storage*, vol. 2, no. 1, pp. 22–40, Feb. 2006.
[15] G. Almási, C. Caşcaval, and D. A. Padua, "Calculating stack distances efficiently," in *Proc. Workshop Memory Syst. Perform. (MSP)*, Berlin, Germany, 2002, pp. 37–43.
[16] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.
[17] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor, "Longest prefix matching using Bloom filters," *IEEE/ACM Trans. Netw.*, vol. 14, no. 2, pp. 397–409, Apr. 2006.
[18] M.-L. Chiang, P. C. H. Lee, and R.-C. Chang, "Managing flash memory in personal communication devices," in *Proc. IEEE Int. Symp. Consum. Electron. (ISCE)*, Singapore, Dec. 1997, pp. 177–182.
[19] L.-P. Chang and T.-W. Kuo, "An adaptive striping architecture for flash memory storage systems of embedded systems," in *Proc. 8th IEEE Real-Time Embedded Technol. Appl. Symp.*, San Jose, CA, USA, Sep. 2002, pp. 187–196.
[20] J. J. Levandoski, P.-A. Larson, and R. Stoica, "Identifying hot and cold data in main-memory databases," in *Proc. IEEE 29th Int. Conf. Data Eng. (ICDE)*, Brisbane, QLD, Australia, Apr. 2013, pp. 26–37.
[21] D. Park, W. He, and D. H. C. Du, "Hot data identification with multiple Bloom filters: Block-level decision vs I/O request-level decision," *J. Comput. Sci. Technol.*, vol. 33, no. 1, pp. 79–97, Jan. 2018.
[22] J. Choi, S. H. Noh, S. L. Min, and Y. Cho, "An implementation study of a detection-based adaptive block replacement scheme," in *Proc. USENIX ATC*, Monterey, CA, USA, 1999, pp. 1–18.
[23] UMass. (2017). *MSR Cambridge Block I/O Traces*. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage
[24] MSR Cambridge. (2017). *OLTP Trace From UMass Trace Repository*. [Online]. Available: http://iotta.snia.org/traces/list/BlockIO
[25] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Trans. Storage*, vol. 4, no. 3, pp. 1–23, Nov. 2008.
[26] M. Russinovich. (2006). *DiskMon for Windows V2.01*. [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/diskmon

HYEONJI HA is currently pursuing the bachelor's degree with the Department of Software, Sookmyung Women's University, Seoul, South Korea.

From 2020 to 2021, she was a Research Assistant with the Big Data Storage Systems Laboratory, under the advice of Prof. Dongchul Park, and studied on dynamic hot data identification with multiple bloom filters and stack distance. Her research interests include big data processing platform, storage systems (NVM, SSD), deep learning, and artificial intelligence.

DAEUN SHIM is currently pursuing the bachelor's degree with the Department of Software, Sookmyung Women's University, Seoul, South Korea.

From 2019 to 2020, she was a Research Assistant with the Big Data Storage Systems Laboratory, under the advice of Prof. Dongchul Park, and did research on implication of Intel Optane Memory for big data processing software platform. Her research interests include big data processing, next generation non-volatile memories (NVM), such as Intel 3D Xpoint memory, back-end system software development (e.g., servers, DB, and API), and DevOps(CICD).



HYEYIN LEE is currently pursuing the bachelor's degree with the Department of Software, Sookmyung Women's University, Seoul, South Korea.

From 2020 to 2021, she was a Research Assistant with the Big Data Storage Systems Laboratory, under the advice of Prof. Dongchul Park, and studied on dynamic hot data identification with multiple bloom filters and stack distance. Her research interests include cloud computing, distributed computing, big data processing and analysis, non-volatile memories (NVM), solid state drives (SSD), and deep learning.

DONGCHUL PARK (Member, IEEE) received the Ph.D. degree in computer science and engineering from the University of Minnesota–Twin Cities, Minneapolis, USA, in 2012.

He was a member of Center for Research in Intelligent Storage (CRIS) Group, under the advice of prof. David H. C. Du. From 2012 to 2016, he was a Senior Research Engineer with the Memory Solutions Laboratory (MSL), Samsung Semiconductor Inc., San Jose, CA, USA, and a Senior Staff Research Engineer with Storage Technology Group (STG) at Intel, Hillsboro, OR, USA, in 2017. From 2017 to 2019, he was an Assistant Professor in computer science and engineering with the Hankuk University of Foreign Studies (HUFS), Yongin, South Korea. Since 2019, he has been an Assistant Professor with the Department of Software, Sookmyung Women's University, Seoul, South Korea. His research interests include storage system design and applications including non-volatile memories (NVM), in-storage computing, data deduplication, key-value store, shingled magnetic recording (SMR) technology, big data processing, Hadoop MapReduce, cloud computing, and next generation NVM, such as Intel Optane Memory.