Sampling-based Garbage Collection Metadata Management Scheme for Flash-based Storage

Biplob Debnath [‡,1] , Srinivasan Krishnan [*,1] , Weijun Xiao [†], David J. Lilja [†], David H.C. Du [†]

[†] University of Minnesota, Minneapolis, USA.
[‡] EMC Corporation, Santa Clara, USA.
[*] Amazon, Seattle, USA.
E-mail: biplob.debnath@emc.com, krishn@amazon.com, wxiao@umn.edu, lilja@umn.edu, du@cs.umn.edu

***Abstract*—Existing garbage collection algorithms for the flash-based storage use score-based heuristics to select victim blocks for reclaiming free space and wear leveling. The score for a block is estimated using metadata information such as age, block utilization, and erase count. To quickly find a victim block, these algorithms maintain a priority queue in the SRAM of the storage controller. This priority queue takes $O(K)$ space, where K stands for flash storage capacity in total number of blocks. As the flash capacity scales to larger size, K also scales to larger value. However, due to higher price per byte, SRAM will not scale proportionately. In this case, due to SRAM scarcity, it will be challenging to implement a larger priority queue in the limited SRAM of a large-capacity flash storage. In addition to space issue, with any update in the metadata information, the priority queue needs to be continuously updated, which takes $O(lg(K))$ operations. This computation overhead also increases with the increase of flash capacity.**

In this paper, we have taken a novel approach to solve the garbage collection metadata management problem of a large-capacity flash storage. We propose a sampling-based approach to approximate existing garbage collection algorithms in the limited SRAM space. Since these algorithms are heuristic-based, our sampling-based algorithm will perform as good as unsampled (original) algorithm, if we choose good samples to make garbage collection decisions. We propose a very simple policy to choose samples. Our experimental results show that small number of samples are good enough to emulate existing garbage collection algorithms.

I. Introduction

With the continually accelerating growth of data, the performance of storage systems is increasingly becoming a bottleneck to improve overall system performance. Many applications, such as transaction processing systems, weather forecasting, large-scale scientific simulations, and on-demand services, are limited by the performance of the underlying storage systems. The limited bandwidth, high power consumption, and low reliability of widely used magnetic disk-based storage systems impose a significant hurdle in scaling these applications to satisfy the increasing growth of data. Flash memory is an emerging storage technology that shows tremendous promise to compensate for the limitations of current magnetic disk-based storage devices. In fact, flash-based Solid State Disks (SSDs) are predicted to be future replacement for the magnetic disk drives. For example, market giants like Dell, Apple, and Samsung have already launched laptops with only SSDs. Microsoft has added features to support SSDs in the new release of Windows 7 operating system. Recently, MySpace.com has switched from using magnetic disk drives in its servers to SSDs as primary storage for its data center operations [1].

Flash memory exhibits different read and write performance characteristics compared to magnetic media. In flash-memory, random read operations are as fast as sequential operations due to lack of mechanical head movement. Whereas, the write operations are substantially slower than the read operations. A distinguishing property of flash memory is that it does not allow overwrite operations. Once data is written in a page, to overwrite it, the page must be erased (which is a slow operation) before writing again. This is known as *in-place update* problem. This problem becomes further complicated because read and write operations are performed in the page granularity, while erase operations are performed in the block granularity (typically a block spans 32-64 pages). Furthermore, a flash memory block can only be erased for a limited number of times [2]. This is known as *wear out* problem.

To hide the limitations of flash memory, a flash-based storage device (for example, SSD) uses Flash Translation Layer (FTL), which helps to acts it like a virtual hard disk drive so that existing disk-based applications can use it without any modifications in the code-base. However, internally FTL needs to deal with the physical properties of flash memory. To solve the *in-place update* problem, FTL writes the updated pages in a log-manner and maintains a logical-to-physical page address mapping table to keep track of the current location of a logical page. FTL periodically invokes garbage collection to perform cleaning operation and reclaim free space. To solve the *wear out* problem, during garbage collection, FTL employs various *wear leveling* algorithms, which spread out the updates uniformly among all blocks so that individual blocks are erased out evenly. For page address mapping and garbage collection, FTL needs to maintain some metadata information. Usually, for the faster access, these metadata are stored on a SRAM-cache in the current flash-based storage device controller [3], [4]. However, due to higher price per byte of SRAM, it is unlikely that SRAM will scale proportionately with the increase of flash capacity [4]. Thus, scaling out

[1] Work done when the author was a graduate student at the University of Minnesota

Block Size	256KB
Metadata Size Per block	8B
Total Blocks Needed for 1GB	4096
Total Metadata Size for 1GB	4096*8B = 32KB
Total Metadata Size for 1TB	1024*32KB = 32MB

TABLE I
METADATA SIZE ESTIMATION

metadata that are stored and processed for a large-capacity flash storage, in the limited amount of SRAM becomes very challenging.

Since SRAM space is a scarce resource, recent research works focus on reducing metadata stored in the SRAM. Most of the research works provide solutions for reducing SRAM space to store the logical-to-physical page address mapping information [3], [4], [5], [6], [7], [8], [9], [10], [11]. For example, recently DFTL [4] proposes a demand based page mapping scheme, which significantly reduces the SRAM space for page mapping [4]. However, very few research works [3], [12] have been conducted to reduce the metadata stored and processed in SRAM for the garbage collection. In this paper, we address this issue by using a sampling-based algorithm.

All the current garbage collection algorithms use score-based heuristics to select a victim block for reclaiming free space and wear leveling. (An overview of these algorithms is given in Section II-A). The block with the highest (or lowest) score is selected as a victim. The score is estimated by using the metadata information, i.e., block utilization, age (time since a block has been lastly erased), or erase count. To implement these algorithms, we need to maintain these metadata per block level. To quickly find a victim block, these algorithms maintain a priority queue and choose a block based on the priority score. The SRAM space for the priority queue is $O(K)$ [13], where K is the flash based storage capacity in terms of total number of blocks. As the flash capacity increases, K increases linearly. For example, Table I shows that for 1GB of flash memory, we need 32KB SRAM to store the metadata, while 32MB for 1TB of flash memory. Clearly, with increase of the flash capacity, scaling out SRAM space for metadata is a problem. Besides the space problem, the priority queue needs to be continuously updated and every update requires $O(lg(K))$ operations [13], whenever any relevant metadata used to estimate the score has been changed, which incurs lot of computation overhead. Current approaches solely focus on solving the SRAM space problem for wear leveling. To reduce the metadata space, they either logically groups adjacent blocks and maintain metadata per group level, or use coding technique to store per block erase count. These approaches are explained in detail in Section II-B. However, these approaches still require $O(K)$ space.

In this paper, we have taken a radically different approach. We propose to use a sampling-based approach to approximate existing garbage collection algorithms. The main idea is as follows. Instead of storing metadata for all K blocks in the SRAM, we store only metadata for N number of randomly selected blocks, where $N << K$. As the time progresses, these samples also change continuously. Whenever we need to select a victim block for garbage collection, instead of considering metadata of all K blocks, we consider only the metadata of the current pool of N sampled blocks. Our sampling-based approximation algorithm significantly reduces the SRAM consumption. Experimental evaluations show that maintaining only small number of samples is good enough for emulating the current garbage collection algorithms. For example, if 30 samples are good enough, then we need only 240 bytes of SRAM space, in contrast to 32MB needed for 1TB flash storage (as shown in Table I). On the other hand, we calculate scores on demand (i.e., during victim selection), thus our approach imposes very less computation overhead.

Here, we do not provide any new garbage collection algorithms, rather we propose a space and computation efficient mechanism to implement the existing garbage collection algorithms in the limited SRAM of a large-capacity flash storage controller. Our algorithm significantly reduces space and computation overhead in the SRAM for the management of garbage collection metadata. To the best of our knowledge, we take the first attempt to reduce both space and computation overhead.

The rest of the paper is organized as follows. Section II describes the background and related works. Section III presents a detailed description of our sampling-based algorithm. Section IV provides an experimental evaluation. Finally, we conclude the paper in Section V.

II. BACKGROUND AND RELATED WORKS

In this section, first we describe the various metadata needed for garbage collection. Next, we describe the related works that reduce metadata information stored in the SRAM for garbage collection.

A. Metadata for Garbage Collection

A garbage collection scheme has two main goals: **(1)** reduce the cleaning cost (i.e., minimize the number of live pages movement from victim blocks) and **(2)** even out wear in different blocks as much as possible. Typically, cleaning and wear-leveling algorithms have conflicting goals. A cleaning-centric policy prefers those blocks which contain small number of valid pages, since they help to minimize the cleaning cost. In contrast, a wear-leveling centric policy chooses the least worn out blocks. These blocks usually store valid (live) pages and contain mostly read-only data. Over the last decades various garbage collection algorithms have been proposed in the literature. All these algorithms require various metadata to select a victim block. In the next subsection, we describe some of these algorithms to give an overview of the different metadata information required to implement them.

1) Cleaning Centric: This category of garbage collection algorithms focus only on cleaning cost. The main goal is to reduce the overhead of cleaning. The greedy strategy is to select a block with the largest number of invalid pages [14]. This strategy works for the uniformly distributed access patterns, while fails for the workloads which exhibit high degree of localities. On the other hand, the cost-benefit (CB) strategy selects a block with the highest $\frac{1-u}{2u} * age$ value, where u stands for the fraction of valid pages in a block (i.e, the ratio

of valid pages and total capacity in terms of pages of a block), and age is the time since last invalidation [14].

2) Wear Leveling: This category of the garbage collection algorithms focus on cleaning efficient policies most of the time, however they conditionally switch back to wear-leveling centric policies. The greedy strategy is to select one of the least worn out blocks. Some algorithms focus on the switching policy: Lofgren et al. proposed that when the difference in erase count between the victim block and the least worn block is more than a threshold (i.e., 15000) wear-leveling centric policy is used [3]. Woodhouse proposed that after every 100-th reclamation, one of the blocks containing only valid data pages will be randomly selected for reclamation [15]. The aim is to move static data to relatively more worn out blocks. Ban proposed that a block will be randomly selected for reclamation after every certain number of reclamations [3]. This interval can be either deterministic or can be randomly determined.

3) Cleaning + Wear Leveling: This category of the garbage collection algorithms consider both cleaning efficiency and wear status during the reclamation decision. Wells proposed a policy that uses a score, which is a weighted average of cleaning cost and wear leveling state of a block [16]. The block with the highest score is selected as a victim. The score of a block j is defined as $score(j) = \alpha * obsolete(j) + (1-\alpha) * max_i\{erasures(i) - erasures(j)\}$, where $obsolete(j)$ stands for the number of invalid pages in block j and $erasure(j)$ stands for the current erase count of block j. The first part of this scoring formula captures the cleaning cost, while the second part captures the impact on wear leveling. The magnitude of the weight determines which part will be given more priority during making the reclamation decision. Usually, $\alpha = 0.8$ is used to calculate this score. Thus, more importance is given to the cleaning cost. However, if the difference between the most and least worn out block becomes more than a threshold (i.e., 500), then $\alpha = 0.2$ is used to give more priority to wear-leveling. On the other hand, Kim et al. defines $score(j) = \lambda(\frac{obsolete(j)}{valid(j)+obsolete(j)}) + (1-\lambda)\frac{erasure(j)}{1+max_i(erasure(i)}$, where λ is monotonic function of the difference between the erase count of the most and least worn out blocks and $0 < \lambda < 1$ [17]. CAT score is defined as $score(j) = \frac{obsolete(j)*age(j)}{valid(j)*erasure(j)}$ [18]. Here, $age(j)$ is the time since the last erasure of unit j, and $\frac{obsolete(j)*age(j)}{valid(j)}$ part works like the cost-benefit (CB) scheme described in Section II-A1, while $\frac{1}{erasure(j)}$ part gives priority to the least worn out blocks.

B. Reducing Space for Garbage Collection Metadata

In this part, we describe the research works that have focused on reducing the metadata space consumption in the SRAM. These algorithms focus on the metadata needed for only wear leveling (i.e., block erase count), while do not consider to reduce computation overhead. In contrast, Our sampling-based approximation algorithm focuses on reducing space as well as computation overhead. In addition, sampling-based algorithm can be used to approximate any existing garbage collection algorithms.

Algorithm 1 Sampling-based Approximation Algorithm

```
1: if (First Iteration) then
2:    Randomly fetch metadata for N fresh blocks from flash memory to SRAM
3: else
4:    Randomly fetch metadata for N − M fresh blocks from flash memory to SRAM
5: end if
6: Calculate scores based on metadata for N sampled blocks
7: Sort the scores in descending (or ascending) order
8: Select the block with max (or min) score as a victim
9: Remove metadata for the victim block from the sample
10: Remove metadata for last N − M − 1 blocks in descending (or ascending) order
```

Group-based wear leveling algorithm [12] uses grouping to save SRAM space. It groups logically adjacent blocks into a single group and maintains an average erase count per group, thus it reduces SRAM space by group size times. However, this scheme is specially designed for the block-level FTL page address mapping. In addition, it still requires $O(K)$ space, where K is the flash capacity in total number of blocks. In contrast our sampling based approach requires $O(N)$ of SRAM space, where $N << K$.

K-Leveling [19] algorithm attempts to minimize the metadata consumption by using coding like technique. It keeps track of the erase count of each block by storing only difference between current erase count and the least worn block's erase count. To store an erase count of 100K, we need at least 20 bits of space. Now, if the difference between current erase count and the least worn block's erase count can be restricted to at most 31, we need only 5 bits to store the erase count information. Thus, by saving only the difference, K-Leveling can reduce the metadata size by four times. However, it requires $O(K)$ space.

III. Sampling-based Algorithm

In this section, first we describe our sampling-based algorithm to approximate existing score-based garbage collection algorithms. Next, we illustrate our proposed algorithm with an example. Finally, we discuss the impact of different sampling parameters.

A. Algorithm

Our sampling-based algorithm maintains garbage collection metadata for a fixed set of sample blocks in the limited SRAM. These samples are selected randomly. We use this sample-pool to select a victim block for garbage collection. Once victim selection is done, we throw out some current samples which are less likely to be selected as victims in the near future. This strategy helps to improve the quality of the sample-pool as the bad samples are thrown out and not considered by the immediate victim selection procedure. In the next iteration, we draw some fresh samples in the SRAM to maintain the fixed size of the sample-pool. This strategy increases the probability of bringing some good samples in the SRAM. Now, we select a victim from the current sample-pool, and so on. The design of our proposed sampling-based algorithm is motivated by the randomized web-cache replacement scheme proposed by Psounis and Prabhakar [20].

Algorithm 1 gives the pseudo-code of our sampling-based victim selection algorithm. At the very beginning (i.e., for the

first time a victim block needs to be selected), we randomly read the metadata of the N blocks from the flash memory (Line 1 in Algorithm 1) to SRAM. Then, we calculate scores (the scores are described in Section II-A) and sort these sampled metadata based on these scores (Line 6-7 in Algorithm 1). Finally, we select a block which has max (or min) score as a victim based on the garbage collection policy used (Line 8 in Algorithm 1). The performance of our proposed sampling-based algorithm depends on the quality of the randomly chosen samples. If the samples are good (i.e., samples that are more likely to be selected as victims), the overall performance is also good. That is why at the end of each iteration, we keep M good samples from the current N samples, and throw out $N - M$ bad samples. We throw out the samples which are less likely to be selected as victim in the next iteration. Out of $N-M$ bad samples, one sample is the current victim block, and rest of the $N - M - 1$ samples are chosen based on the score. We throw out the last $N - M - 1$ samples in sorted order as their scores indicate they have very low (or zero) probability of being selected as victim in the next iteration (Line 9-10 in Algorithm 1). When we need to perform the next victim selection, we again randomly read $N - M$ fresh samples from the flash to SRAM (Line 4 in Algorithm 1), and make eviction decision based on the metadata of these $N-M$ fresh samples and previously retained M good samples. This procedure is repeated whenever we need to select a victim block.

B. An Illustration

To illustrate our sampling-based approach, we assume our victim selection criteria prefers a block with the minimum erase count as a candidate for garbage collection. We name this as *Greedy_Wear* scheme. Now, we will approximate *Greedy_Wear* scheme by keeping only metadata for $N = 5$ blocks in the SRAM. At end of each iteration, we retain two good samples and throw out three samples, i.e., $M = 2$. At first, we randomly select metadata for five fresh sample blocks from the flash memory and bring them to SRAM. The block erase count of the samples are shown in Figure 1(a). Next, we sort the samples based on the erase count as shown in Figure 1(b). We select the block having the minimum erase count (i.e., 10) as victim and remove it metadata from SRAM. Now, we need to remove $N - M - 1 = 5 - 2 - 1 = 2$ more samples from SRAM. Since, our victim selection criteria is the block having minimum erase count, we remove last two blocks in the ascending order of erase counts. The victim block is marked in the dashed rectangle and all the removed blocks are marked in gray color, as shown in Figure 1(c). After removing these three samples, currently we have only two metadata sampled blocks in SRAM, as shown in Figure 1(d). During the next victim selection process, again we randomly select metadata for three fresh sample blocks from flash memory and bring them to SRAM. These fresh samples (marked in light gray color) and two previously retained samples are shown in Figure 1(e). Now, from these five samples, we select the block having erase count of seven as a victim, and so on.

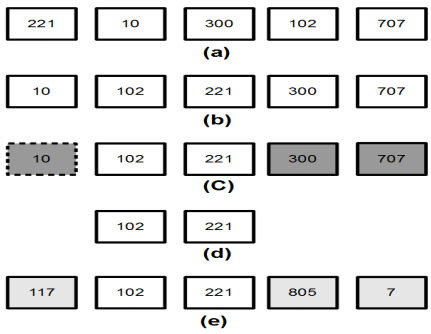


Fig. 1. Illustration of sampling-based approximation algorithm for $N = 5$ and $M = 2$. Here, the block with minimum erase count is selected as a victim.

C. Parameter Selection

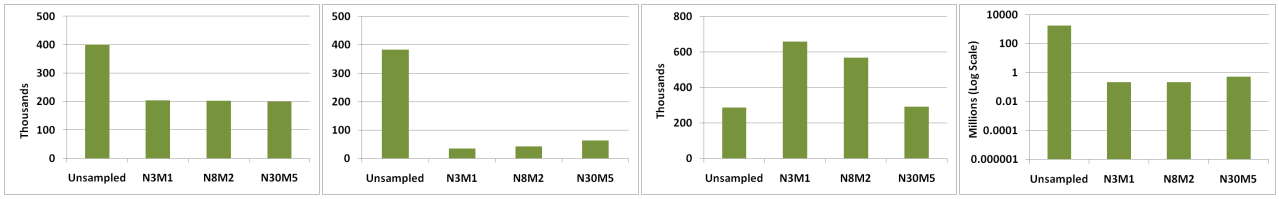
Usually, smaller values of M help to choose better victim candidates. The main reason behind this behavior is that for a fixed value of N, as the value of M increases, fewer number (i.e., $N - M$) of fresh samples are drawn from the flash memory, while larger number of old samples (i.e., M) are retained in the SRAM in order to make the eviction decision. As a result, performance degradation occurs due to the relatively large number of bad samples. On the the other hand, the value of $N - M$ contributes to the number of times that we need to perform metadata read operations from the flash memory. Since, flash read operations are much slower than the SRAM access operations, we need to keep $N-M$ as small as possible in order to quickly perform victim selection operation. However, it may have an adverse effect on the performance of the sampling-based algorithm.

IV. Experimental Evaluation

In this section, we study the performance of our sampling-based approximation algorithms by using real-world trace driven simulation. We emulate existing garbage collection schemes by using our sampling-based algorithms, and compare their performance according to various garbage collection evaluation metrics.

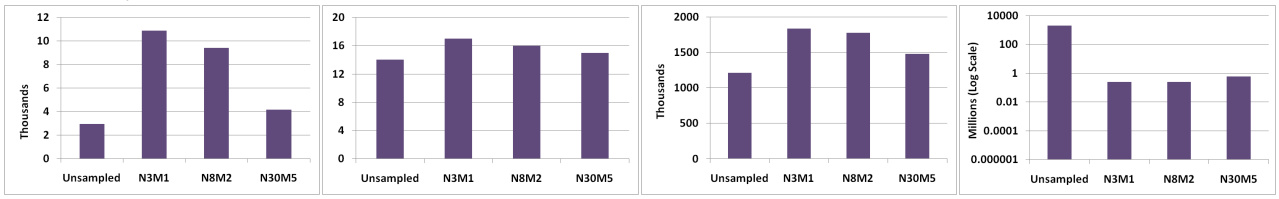
A. Experimental Setup

To evaluate the performance of our sampling-based schemes, we have used Solid State Drive (SSD) simulator developed by The Pennsylvania State University [21]. In our evaluation, DFTL [4] is used as underlying FTL page address mapping scheme. We have used three enterprise class real-world traces to study the impact of the sampling-based algorithms. For the lack of space, here we only include the results of write-intensive Financial-1 trace from the UMass Storage Repository [22]. We have used the following three sampling settings: (a) $N = 3, M = 1$, (b) $N = 8, M = 2$, and (c) $N = 30, M = 5$. We refers these three settings as **N3M1, N8M2**, and **N30M5**, respectively. The setting **N3M1** refers that we maintain only $N = 3$ samples in SRAM to select a victim block for garbage collection, and once a victim is



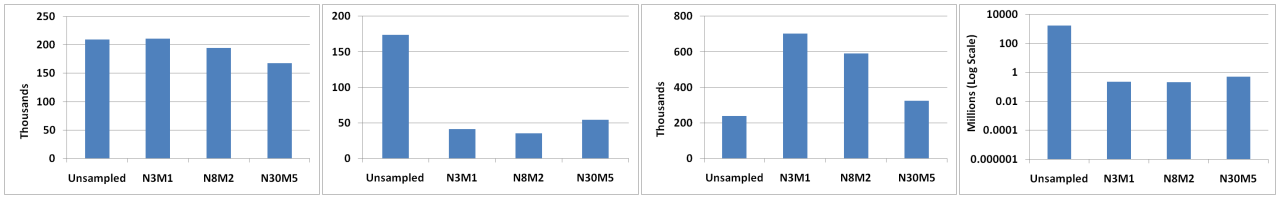
(a) Variance of Erase Counts (b) Max Erase Count (c) GC Overhead (d) Victim Selection Overhead

Fig. 2. Greedy_Clean Scheme for Financial-1 (2GB)



(a) Variance of Erase Counts (b) Max Erase Count (c) GC Overhead (d) Victim Selection Overhead

Fig. 3. Greedy_Wear Scheme for Financial-1 (2GB)



(a) Variance of Erase Counts (b) Max Erase Count (c) GC Overhead (d) Victim Selection Overhead

Fig. 4. CB Scheme for Financial-1 (2GB)

selected, we retain only $M = 1$ current sample for the next iteration. During next iteration, we draw $N - M = 3 - 1 = 2$ fresh samples from the flash memory. For the detail experimental setup and result analysis, the readers are referred to the technical report of this paper [23].

B. Result Analysis

In this section, for the lack of space, first we discuss the results for the Financial-1 workload [22]. Next, we summarize the results for two other real-world traces.

1) Financial-1 (2GB): **Greedy_Clean Policy.** This policy selects a block with the largest number of invalid pages as a garbage collection victim. In terms of *erase counts variance* and *max erase count*, sampling-based algorithm works better than the unsampled (original) algorithm. Figure 2(a) shows that *variance* is almost same for all three sampling settings, while Figure 2(b) shows that *max erase count* becomes higher with the increase of sample size. At a first glance, these results seem counter-intuitive, as it is expected that unsampled version will always outperform sampling-based version, after all it is an approximation of the unsampled one. However, these results are correct. The main reason is as follows. Greedy_Clean scheme is a cleaning efficient policy and it does not consider wear leveling factor at all. *Erase count variance* and *max erase count* metrics are biased to wear leveling. Sampling-based algorithm does not consider enough information (i.e., does not consider all blocks metadata) to make 100% cleaning effcient decision as it is an approximation (i.e, makes decisions from the metadata of a small number of sampled blocks), which makes it inherently biased to the wear leveling metrics. That is why unsampled version performs worse for these two metrics. With more samples, the sampling-based algorithm becomes closer approximation of the unsampled one, and that is why with an increase in sample size, *max erase count* becomes larger and difference between *max erase count* values of the unsampled and sampling-based algorithm decreases.

Figure 2(c) shows that in terms of *garbage collection overhead*, unsampled version performs better. As expected, with an increase in sample size, sampling-based algorithm exhibits performance similar to the unsampled one. These results are intuitive, since Greedy_Clean scheme is specially optimized for garbage collection overhead, quite naturally it will perform better. The bigger is the sample size, the better is the approximation, and the closer is the performance. Figure 2(c) shows that for **N30M5** i.e., $N = 30$ and $M = 5$, sampling-based algorithm performs almost as good as unsampled one.

Figure 2(d) shows that when metadata is not stored in the SRAM, to select a victim, the overhead in terms of the total number of flash read operations. This overhead is several magnitude lesser for the sampling-based algorithm compared to unsampled one. This is expected, as sampling case, we need to read only fewer number of sampled blocks, while unsampled algorithm scans all blocks to select a victim. As expected, in the sampling-based algorithm, as the $N - M$ (i.e., number of fresh blocks that needs to be read) increases, the overhead also increases proportionately. Overall, $N = 30$ and $M = 5$ shows better performance compared to the other sampling settings.

Greedy_Wear Policy. This policy selects the least worn out block as a victim. In terms of *erase counts variance* and *max erase count*, sampling-based algorithm performs worse than the unsampled case. However, Figure 3(a) and Figure 3(b) show that as the sample size increases, *variance* and *max erase count* become closer to the unsampled one. This results are quite expected as Greedy_wear is a wear leveling centric policy, and these two metrics biased to wear leveling. That is why unsampled version outperforms sampled version. With more samples (i.e., $N = 30$), sampling-based algorithm becomes closer approximation of the unsampled case, and exhibits closer performance. Figure 3(c) shows that in terms of *garbage collection overhead* unsampled version performs better. As expected, again with the more number of samples, sampling-based algorithm approximates better, and exhibits better performance. Figure 3(d) shows that with the increase of $N-M$, the victim selection overhead becomes larger. Overall, $N = 30$ and $M = 5$ setting shows better performance.

CB Policy. The Cost-benefit (CB) scheme selects a block with the highest utility value of $\frac{1-u}{2u} * age$ (as described in Section II-A) as a garbage collection victim [14]. Similar to the Greedy_Clean algorithm, the CB algorithm is also a cleaning centric scheme, while it considers more metadata information to select victims. In terms of *erase count variance* and *max erase count*, sampling-based algorithm performs better than the unsampled one. The reasons are same as the case of Greedy_Clean scheme explained earlier. In terms of *garbage collection overhead* and *victim selection overhead*, Figure 4(c) and Figure 4(d), respectively, show that CB scheme exhibits similar trends as the Greedy_Clean scheme. Note that, $N = 30$ and $M = 5$ setting exhibits overall better performance.

2) Result Summary for Other Traces: In addition, to Financial-1 (2GB) trace, we evaluate sampling-based algorithms for the Financial-1 (4GB), Financial-2 (2GB), and Microsoft Cambridge Labs (8GB) block I/O traces (more detail could be found in [23]). These traces also show similar trends as Financial-1 (2GB) workload. Overall, we find that $N = 30$ and $M = 5$ setting can better approximate existing garbage collection algorithms.

V. Conclusion

This paper proposes a sampling-based (i.e., randomized) approach to approximate existing garbage collection algorithms. Our approach is very easy to implement and it takes very less space in SRAM as well as incurs less computation overhead. Our preliminary experimental results show that sampling-based approximation algorithms are wear leveling friendly and they scale very well for the large-capacity flash-based storage. For the three real-world traces, we find that small number of samples (i.e., $N = 30$ and $M = 5$) can emulate existing garbage collection algorithms. Moreover, the number of samples needed for approximation do not increase with the increase in flash capacity (i.e., 2GB, 4GB, and 8GB). Thus, our sampling-based approach looks very promising. However, we need to perform more analysis and experiments in order to verify and validate that small number of samples can work for any size of flash capacity. We will address this issue in the future.

Acknowledgment

We would like to thank Ludmila Cherkasova for pointing out the randomized approach to implement a priority queue. This work was supported in part by the Center for Research in Intelligent Storage (CRIS), which is supported by National Science Foundation grant no. IIP-0934396 and member companies, and the National Science Foundation under grant no. ITR-0937060 to the Computing Research Association for the CIFellows project.

References

[1] White Paper: MySpace Uses Fusion Powered I/O to Drive Greener and Better Data Centers. http://www.fusionio.com/PDFs/myspace-case-study.pdf.

[2] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," in *USENIX*, 2008.

[3] E. Gal and S. Toledo, "Algorithms and Data Structures for Flash Memories," *ACM Computing Survey*, vol. 37, no. 2, 2005.

[4] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings," in *ASPLOS*, 2009.

[5] S. Lee, D. Park, T. Chung, D. Lee, S. Park, and H. Song, "A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation," *ACM Trans. Embed. Comput. Syst.*, vol. 6, no. 3, 2007.

[6] S. Lee, D. Shin, Y. Kim, and J. Kim, "LAST: Locality-Aware Sector Translation for NAND Flash Memory-Based Storage Systems," in *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 6, 2008.

[7] J. Kim, J. Kim, S. Noh, S. L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for Compact Flash Systems," in *IEEE Transactions on Consumer Electronics*, vol. 48, no. 2, 2002.

[8] J. Kang, H. Jo, J.-S. Kim, and J. Lee, "A Superblock-based Flash Translation Layer for NAND Flash Memory," in *EMSOFT*, 2006.

[9] C. Wu and T. Kuo, "An Adaptive Two-Level Management for the Flash Translation Layer in Embedded Systems," in *ICCAD*, 2006.

[10] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J. Kim, "A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND Flash-Based Applications," *ACM Trans. Embed. Comput. Syst.*, vol. 7, no. 4, 2008.

[11] D. Park, B. Debnath, and D. Du, "CFTL: A Convertible Flash Translation Layer with Consideration of Data Access Patterns," *UMN CS Technical Report*, no. TR 09-023, 2009.

[12] D. Jung, Y. Chae, H. Jo, J. Kim, and J. Lee, "A Group-based Wear-Leveling Algorithm for Large-Capacity Flash Memory Storage Systems," *CASES*, 2007.

[13] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. McGraw-Hill Higher Education, 2002.

[14] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," in *USENIX*, 1995.

[15] D. Woodhouse, "JFFS: The Journalling Flash File System," *Proceedings of Ottawa Linux Symposium*, 2001.

[16] S. Wells, "Method for Wear Leveling in a Flash EEPROM Memory," *United States Patent*, no. 5341339, 1994.

[17] H. Kim and S. Lee, "An Effective Flash Memory Manager for Reliable Flash Memory Space Management," *IEICE Transactions on Information and Systems*, vol. E85-D, no. 6, 2002.

[18] M. Chiang, P. Lee, and R. Chang, "Cleaning Policies in Mobile Computers Using Flash Memory," *Journal Systems and Software*, vol. 48, no. 3, 1999.

[19] S. Park, "K-Leveling: An Efficient Wear-Leveling Scheme for Flash Memory," in *UKC*, 2005.

[20] K. Psounis and B. Prabhakar, "Efficient Randomized Web-Cache Replacement Schemes Using Samples From Past Eviction Times," *IEEE/ACM Transaction on Networking*, vol. 10, no. 4, 2002.

[21] "A Simulator for Various FTL Schemes," http://csl.cse.psu.edu/?q=node/322.

[22] "University of Massachusetts Amhesrst Storage Traces," http://traces.cs.umass.edu/index.php/Storage/Storage.

[23] B. Debnath, S. Krishnan, W. Xiao, D. Lilja, and D. Du, "Sampling-based Metadata Management for Flash Storage," *UMN ECE Technical Report*, no. ARCTiC 10-01, 2010.