Impact of Data Locality on Garbage Collection in SSDs: A General Analytical Study

Yongkun Li[†], Patrick P. C. Lee[‡], John C. S. Lui[‡], Yinlong Xu[†]
[†]School of Computer Science and Technology, University of Science and Technology of China
[‡]Department of Computer Science and Engineering, The Chinese University of Hong Kong
[†]{ykli,ylxu}@ustc.edu.cn, [‡]{pclee,cslui}@cse.cuhk.edu.hk

ABSTRACT

Solid-state drives (SSDs) necessitate garbage collection (GC) to erase data blocks and reclaim the space of invalidated data, and GC inevitably introduces additional writes due to data relocation. The performance of GC, which is quantified by cleaning cost or write amplification, is critical to the overall performance of SSDs. However, characterizing GC performance is complicated by the general implementations of GC algorithms and the complex data locality characteristics of real-world workloads. This paper presents a *general* analytical study to characterize the performance impact of data locality on a general family of GC algorithms. We develop probabilistic models to address two fundamental issues: (1) What is the impact of data locality on the performance of locality-oblivious GC? (2) How can data locality be leveraged to improve the performance in locality-aware GC? We further conduct extensive trace-driven simulations on real-world workloads to validate the findings of our models.

Categories and Subject Descriptors

B.3.3 [**Memory Structures**]: Performance Analysis and Design Aids; D.4.2 [**Operating Systems**]: Storage Management—*Garbage collection*; G.3 [**Probability and Statistics**]: Probabilistic algorithms

General Terms

Performance;Theory;Algorithms

Keywords

SSDs; Garbage Collection; Trade-off; Data Locality

1. INTRODUCTION

NAND-flash-based solid-state drives (SSDs) provide performance gains over traditional hard-disk drives (HDDs) in I/O throughput, energy efficiency, and reliability, and thus have seen wide adoption in mainstream storage systems. SSDs have distinct I/O characteristics from HDDs. An SSD organizes data into *blocks*, each containing a number of fixed-size *pages*. Reads and writes are performed on pages. To write new data, an SSD must identify a *clean* page to program and mark the programmed page as *valid*; to update existing data, an SSD uses an *out-of-place* approach that first programs the data to a new clean page and then marks the original page as *invalid*. Invalid pages can only be reset to clean pages by an erase operation, which must be performed on the whole block. Thus, an SSD regularly performs *garbage collection* (GC) to erase blocks so as to reclaim the space of invalid pages. A physical constraint is that each SSD block can only tolerate a finite number of erase operations before wearing out.

The design of a GC algorithm, which specifies the strategy of choosing a block to erase, determines the performance of an SSD. Specifically, a GC algorithm relocates all valid pages to a different free block with clean pages and hence incurs additional writes. Its performance, often quantified as *cleaning cost* [17] or *write amplification* [12], affects the performance of normal read/write operations in the foreground. Since an SSD block can wear out, a GC algorithm is often designed to improve GC performance (e.g., by erasing the block with the fewest valid pages), while avoiding over-wearing a particular SSD block. It is shown that the trade-off between performance and durability can be realized via different parametric configurations of a general family of GC algorithms [17].

The analysis of general GC algorithms becomes challenging when taking into account the heterogeneity of storage workloads. Specifically, real-world storage workloads are known to exhibit high *data locality*, where some stored data is frequently accessed (i.e., hot), while the rest is rarely accessed (i.e., cold) [23]. It is a common consensus that separating the placement of hot/cold data can improve GC performance [11, 14]. However, general non-uniform workloads can be more complex and exhibit different degrees of hotness/coldness. Their implications on different implementations of GC algorithms remain an open issue. To characterize the GC performance of SSDs, measurement studies (e.g., trace-driven simulations) can be used, yet they suffer from the efficiency problem due to the variety of workloads and the wide choices of GC implementations. On the other hand, analytical modeling is easy to be parameterized and generally needs less running time. However, analytical studies on GC in the literature (see §7) often lack generality, and do not explicitly address the relationships between different types of workloads and different implementations of GC algorithms.

ICPE'15, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.

http://dx.doi.org/10.1145/2668930.2688036.

In this paper, we propose a general analytical model that formally characterizes the impact of data locality on the cleaning performance of a general family of GC algorithms, with emphasis on taking into account the generality of both data locality characteristics and implementations of GC algorithms. We make the following contributions:

- We first formalize the notion of data locality. From the study of real-world traces, we observe that real-world workloads exhibit the properties of *clustering* (i.e., only a small proportion of address space is accessed) and *skewness* (i.e., accesses to different parts of address space vary significantly). We formulate a general workload model that classifies the regions of address space into different types of access frequencies, so as to capture both the clustering and skewness.
- We present probabilistic models that analyze the cleaning behaviors for *locality-oblivious GC*, whose implementation builds on a family of GC algorithms called the *Greedy Random Algorithm (GRA)*. GRA can achieve the performance of GREEDY, RANDOM and those in between through a tunable parameter. We formally show that data locality may severely degrade GC performance, and the influence becomes more significant when GRA moves from RANDOM to GREEDY.
- We further extend our model to study the performance of *locality-aware GC*, which incorporates data locality information into the GC design. Based on the knowledge that data pages have different types of access frequencies, we analyze the design strategy of *data grouping*, which stores different types of data pages in separate regions of an SSD.
- We conduct extensive trace-driven simulations using the SSD simulator [1]. We validate the accuracy of our model in performance characterization. We evaluate both locality-oblivious GC and locality-aware GC through several real-world traces from different storage environments, and confirm our findings derived from our analytical model. We also study the performance-durability trade-off of GC algorithms via trace-driven simulations.

The remaining of the paper proceeds as follows. In §2, we formulate the problem of analyzing the impact of data locality on GC performance. In §3, we present probabilistic analysis on the cleaning behaviors of locality-oblivious GC, and study the impact of data locality. In §4, we focus on locality-aware GC, and study the impact of data grouping. In §5, we validate our analysis via trace-driven simulations and identify the implications of our model. In §6, we present trace-driven simulation results under real-world traces. In §7, we review related work, and finally in §8, we conclude the paper.

2. PROBLEM FORMULATION

In this section, we formulate the problem of analyzing the impact of data locality on the performance of GC algorithms. We first present an SSD model. Based on the analysis of real-world storage traces, we then propose a new model that can capture the data locality of workloads. Finally, we define the metrics for quantifying the performance of GC algorithms.

2.1 SSD Model

We consider an SSD with N physical blocks with k pages each. Recall from §1 that a block consists of a combination of valid, invalid, and clean pages. Since an SSD reserves blocks for GC and bad block management [19], its advertised capacity as viewed by the logical address space is generally smaller than its physical capacity. We define the proportion of reserved blocks as the *spare factor* S. That is, NS blocks are reserved, and we call them *spare blocks*. Since the logical address space only spans $N(1-S)$ blocks, increasing S implies less available storage space, yet it reduces write amplification since each block contains fewer valid pages on average. Thus, S is a critical parameter to GC performance.

To support out-of-place updates (see §1), an SSD maintains address mappings between logical and physical pages in a software layer called *flash translation layer (FTL)*. In this work, we assume that address mapping is implemented in the *page level* since it potentially achieves the best I/O performance [9], although it can also be realized in the block level [22] or hybrid form [6, 15, 21]. Note that page-level address mapping is also assumed in prior analytical work [3, 7, 12, 26, 27].

With page-level address mapping, we consider the following write and GC implementations, which are also used by previous analytical studies (see §7). At any time, there is one special block called the *write frontier*. All page writes, such as external writes (due to workload) and internal writes (due to GC), are directed to the write frontier, and pages are sequentially written to the write frontier. When all clean pages in the write frontier are used up, the write frontier is *sealed* and no more writes are permitted until a clean block is selected as the write frontier. Thus, an SSD contains three types of blocks at any time: (1) the write frontier (containing valid, invalid, and clean pages), (2) sealed blocks (without any clean pages), and (3) clean blocks (with clean pages only). When the number of clean blocks drops below a certain threshold, GC will be triggered to perform the following steps: (1) select a sealed block, (2) write all valid pages from the selected sealed block to the write frontier, and (3) erase the selected block. To fully utilize the available spare blocks, we set the threshold as small as possible, so GC is triggered only when the SSD no longer contains any clean physical block after the write frontier is allocated, and GC stops when at least one clean block is reclaimed.

2.2 Workload Model

We propose a workload model to characterize data locality in real-world storage workloads. To motivate the characterization, we first analyze eight real-world I/O traces collected from different storage environments (see §6.1 for details). Since read requests do not affect the GC performance, we focus on write requests only, and accessing a page means that the data in this page is updated. We configure the volume size of an SSD as the maximum logical page number being accessed, and measure the distribution of the access frequency of logical pages for each I/O trace. Here, the access frequency of a logical page is defined as the total number of writes to this page during the whole period of executing a workload. Figure 1 depicts the distributions of four traces, while the remaining traces also share similar trends (which are not shown in the interest of space). We make two observations. First, if we measure the proportion of pages which have *at least one access*, it only accounts for a small por-

tion of the entire logical address space. For example, for the Financial, Webmail, Online, and Webmail+Online traces, the proportions of accessed pages are only 12.4%, 4.6%, 3.3%, and 5.2%, respectively. This reflects a high degree of *clustering* in the workloads. Secondly, among the accessed pages, the number of times that each page is accessed varies significantly. For example, the highest access frequencies of pages in these four traces are around 9.1×10^4, 3.8×10^4, 3.1×10^4, and 6.0×10^4, respectively, implying that some pages are re-accessed frequently. This reflects high *skewness* of the workloads.

We formulate our workload model based on our trace analysis as follows. To model the clustering property, we define a proportion f_a of the logical address space that gets accessed as the *active region*, while the remaining logical address space as the *inactive region*. The active (inactive) region contains all active (inactive) logical pages. With page-level address mapping, each active (inactive) logical page is mapped to an active (inactive) physical page. In our analysis, we do not distinguish the physical and logical pages explicitly, and we simply refer to them as active or inactive pages. Thus, the number of active pages over an SSD is $N(1-S)kf_a$.

Since a real workload usually accesses only a very small proportion of the address space of the SSD, to model the clustering property, we let the active and inactive pages be mapped to separate blocks in the physical address space. That is, the region of the active pages can be taken as the working set of the workload. We call the blocks containing only inactive pages *inactive blocks*, and they must be sealed as they are also occupied by valid data which never gets updated during the whole period of executing the workload. We call the remaining sealed blocks *active blocks*, which may contain both valid and invalid pages.

To model the skewness property, we further classify the active pages into n *access types*. Let $\mathbf{r} = (r_1, ..., r_n)$ and $\mathbf{f} = (f_1, ..., f_n)$ be the vectors, such that type-i (where $1 \leq i \leq n$) pages are uniformly accessed by a proportion r_i of requests which account for a proportion f_i of active pages of the logical address space, and we have $\sum_{i=1}^{n} r_i = \sum_{i=1}^{n} f_i = 1$. Without loss of generality, we assume that $\frac{r_1}{f_1} \geq \frac{r_2}{f_2} \geq \cdots \geq \frac{r_n}{f_n}$. A larger ratio $\frac{r_i}{f_i}$ means that a type-i page is on average accessed by more requests, so our assumption implies that type-1 pages are the hottest, while type-n pages are the coldest. To summarize, we can characterize a workload by the parameters $(f_a, n, \mathbf{r}, \mathbf{f})$, as illustrated in Figure 2.

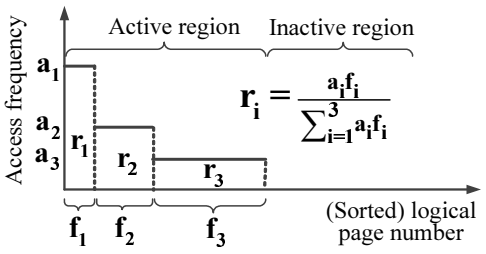


Figure 2: Illustration of our workload model with $n = 3$.

We show via examples how to map a workload to our model. The uniform workload can be modeled by setting $f_a = 1, \mathbf{r} = (1)$ and $\mathbf{f} = (1)$. A non-uniform hot/cold workload [7,27], such as 80/20 workload, can be modeled by setting $f_a = 1$, $n = 2$, $\mathbf{r} = (0.8, 0.2)$, and $\mathbf{f} = (0.2, 0.8)$. We can also model a more fine-grained workload by setting a larger n. For instance, for the Webmail+Online workload in Figure 1(d), the proportion of active pages f_a is 0.052, and the embedded figure in Figure 1(d) shows the distribution of access frequency of active pages only. We can choose $n = 3$ and set the thresholds as 20 and 5. That is, pages that are accessed by no less than 20 times are classified as type-1, other pages that are accessed by no less than 5 times are classified as type-2, and the remaining active pages are of type-3. Under this setting, the parameters $\mathbf{r}$ and $\mathbf{f}$ can be measured as $\mathbf{r} = (0.946, 0.036, 0.018)$ and $\mathbf{f} = (0.270, 0.135, 0.595)$.

Note that our workload model differs from the traditional **r–f** model in [23] (which is also used in prior analytical studies [7,27]) in that the latter does not consider the clustering property (i.e., it sets $f_a = 1$). Our workload model can be viewed as a generalized $\mathbf{r}-\mathbf{f}$ model in [23], and it enables us to model a wider range of real-life I/O workloads.

Our analysis considers the entire period of executing a workload. Note that we focus on write requests only, we let L be the total number of single-page writes in the entire workload. To guarantee the statistical significance, we assume that L is sufficiently larger than the capacity of the logical address space, i.e., $L \gg Nk(1-S)$.

2.3 Performance Metric

Given the SSD and workload models, our analysis focuses on the metric *cleaning cost* $\mathcal{C}(alg)$, which quantifies the performance of a GC algorithm alg. Let M be the total number of GC operations that have been performed on an SSD, v_j (where $1 \leq j \leq M$ and $0 \leq v_j \leq k$) be the number of valid pages moved from the erased block to the write frontier in the j-th GC operation. Mathematically, we define the cleaning cost as the total number of internal page writes in all GC operations. That is,

$$\mathcal{C}(alg) = \sum_{j=1}^{M} v_j. \quad (1)$$

Note that the cleaning cost equivalently models write amplification [12], which is often defined as the average number of internal page writes per external user write.

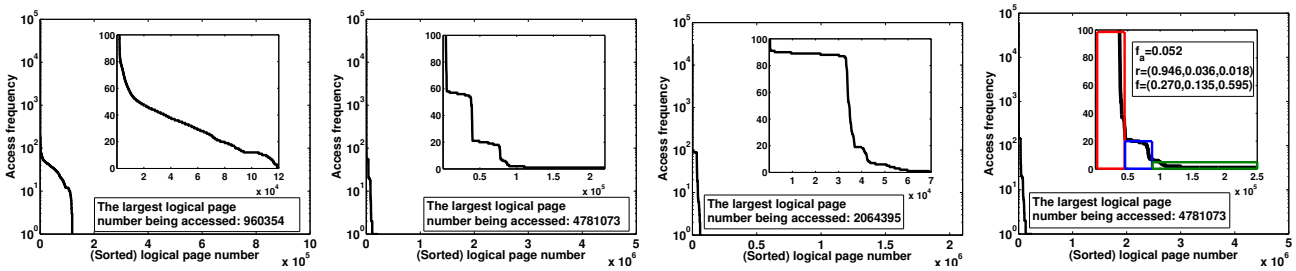
In this paper, we study two classes of GC designs, namely *locality-oblivious GC* and *locality-aware GC*. Locality-oblivious GC does not consider data locality in the design. We analyze how data locality influences the cleaning cost of GC algorithms under this category. See §3 for details. Locality-aware GC aims to include the workload characteristics defined by $(f_a, n, \mathbf{r}, \mathbf{f})$ into the design. We consider the design strategy of *data grouping*, which stores the data pages of different access types separately in an SSD. See §4 for details.

3. LOCALITY-OBLIVIOUS GC

In this section, we propose a *probabilistic analysis framework* on a family of locality-oblivious GC algorithms. The framework enables us to quantify the cleaning cost. We then study some special cases of GC algorithms.

3.1 General Analysis Framework

We consider a family of locality-oblivious GC algorithms, which we call the *Greedy Random Algorithm (GRA)*. The operations of GRA are defined by a *window size* parameter d (where $1 \leq d \leq N$): each time when GC is triggered, we first select d sealed blocks that have the smallest number of valid



(a) Financial (b) Webmail (c) Online (d) Webmail+Online

Figure 1: Distributions of access frequency of logical pages for four different real-world workloads. The logical page numbers in the x-axis are sorted in descending order of access frequencies.

pages as candidates (random tie-breaking is used if multiple blocks contain the same number of valid pages); then we uniformly select a block from these d candidate blocks for GC. We use d to denote an implementation of a GC algorithm when the context is clear. In particular, if $d = 1$, then GRA always selects the sealed block containing the smallest number of valid pages for GC, and we call it the *GREEDY* algorithm. On the other hand, if $d = N$, then GRA uniformly selects a sealed block among all physical blocks, and we call this special GRA the *RANDOM* algorithm. We see that GREEDY locally minimizes the cleaning cost as it always write the fewest valid pages in each GC, while RANDOM maximizes the evenness of erasures on blocks as it in essence has all blocks evenly erased. Therefore, GREEDY and RANDOM represent the two *extreme points* in the design space of GC algorithms [17], and GRA operates at the points between GREEDY and RANDOM via the tunable window size parameter.

We now describe the dynamics of the physical blocks in locality-oblivious GC. Figure 3 depicts the states of physical blocks in an SSD right after a clean block is allocated as the write frontier. Recall that GC is triggered when the SSD has no clean block left (see §2.1). Both active blocks and inactive blocks may be chosen for GC according to d: if an inactive block is selected, then k additional page writes are always required as the block contains all valid pages that are not updated; otherwise, if an active block is chosen for GC, then the number of additional page writes is a random variable. Let $\overline{C}_a(d)$ be the average number of all valid pages in the block, and $\overline{C}_i(d)$ be the average number of type-i (where $1 \le i \le n$) valid pages in the block. We have $\overline{C}_a(d) = \sum_{i=1}^{n} \overline{C}_i(d)$.

Our GRA analysis makes the following approximation: the d candidate blocks are chosen from the d blocks that are sealed in the earliest time, since the earlier sealed blocks should contain fewer valid pages on average. We note that all writes are designated to the unique write frontier without differentiating the type of updated pages in the case of locality-oblivious GC. Suppose that $n \ll k$, which is expected since we have $k \ge 64$ in commodity SSDs and n is small. Then every sealed block should likely contain *all* n types of pages (see Figure 3). Since our workload model assumes that all pages of the same access type have the same probability of being accessed and updated, we expect that an earlier sealed block has more pages of each type invalidated and hence fewer valid pages overall. In particular, GREEDY can be approximated by the first-in-first-out (FIFO) algorithm, which reclaims blocks following their order of being sealed.

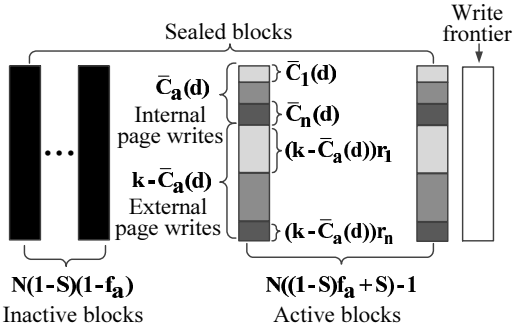


Figure 3: States of blocks right after a clean block is allocated as the write frontier.

In the following, we analyze the cleaning cost of GRA for general window size d. We refer readers to Appendix for the proofs of all theorems in this paper.

THEOREM 1. *The cleaning cost of GRA with window size d is*

$$\mathcal{C}(d) = \big[L/(k - \overline{C}(d))\big]\overline{C}(d), \quad (1 \le d \le N), \tag{2}$$

where $\overline{C}(d)$ denotes the average cleaning cost in each GC and it is

$$\overline{C}(d) = \begin{cases} \overline{C}_a(d), & \text{if } d \le N_a, \\ (N_a/d)\overline{C}_a(d) + (1 - N_a/d)k, & \text{otherwise,} \end{cases} \tag{3}$$

where $N_a = N((1-S)f_a + S) - 1$ denotes the number of active blocks. As $N \to \infty$, $\overline{C}_a(d)$ can be computed via Equations (4) and (5) if $1 \le d \le N_a$, and $\overline{C}_a(d) = \overline{C}_a(N_a)$ if $d > N_a$.

$$\overline{C}_a(d) = \sum_{i=1}^{n} \overline{C}_i(d), \tag{4}$$

$$\overline{C}_i(d) = \frac{r_i(k - \overline{C}_a(d))\,(1 - P')^{N_a - d}}{1 + P' \times d - (1 - P')^{N_a - d}}, \tag{5}$$

where $P' = \frac{r_i(k - \overline{C}_a(d))}{(N_a+1)(1-S')kf_i}$ and $S' = \frac{S}{(1-S)f_a+S}$.

Remarks: S' denotes the spare factor of the active region and P' represents the average conditional probability that an active block has a type-i page invalidated for each external page write given that it accesses a type-i page. Equation (5) shows that *the cleaning cost of GC may be affected by*

data locality, including both clustering and skewness, which can be reflected by the parameters f_a, r_i's and f_i's. We will further discuss this impact for different values of d in more detail in the following subsections. ∎

Theorem 1 provides a general framework to derive the cleaning cost of GRA for general window size d. However, Equation (5) in Theorem 1, which characterizes the average number of type-i valid pages containing in each active block to be reclaimed, i.e., $\overline{C}_i(d)$, is still too complicated to solve. Since d varies from one to infinity as N goes to infinity, to further simplify the equation so as to efficiently derive the cleaning cost of GRA, we consider the following cases for d: (1) $d = o(N)$, i.e., $\lim_{N\to\infty} \frac{d}{N} = 0$, (2) $d \geq N_a$, and (3) $d = \alpha N_a$ ($0 < \alpha < 1$) where α is a constant. Note that GREEDY ($d = 1$) and RANDOM ($d = N$) are included in the first two cases. In the following, we focus on the average cleaning cost in each GC, i.e., $\overline{C}(d)$, because the cleaning cost can be derived based on Theorem 1 given $\overline{C}(d)$.

3.2 GRA with Window Size $d = o(N)$

Consider the case where $d = o(N)$, which also includes the scenario of GREEDY ($d = 1$). Equation (5) can be further simplified when N goes to infinity.

THEOREM 2. *As $N \to \infty$, the average cleaning cost $\overline{C}(d)$ of GRA with window size $d = o(N)$ is the unique solution of Equation (6) in the range $[0, k]$*[1].

$$\overline{C}(d) = \sum_{i=1}^{n} \big[(k - \overline{C}(d))r_i\big] \,/\, \Big[e^{A_i} - 1\Big], \tag{6}$$

where $A_i = \frac{r_i(k-\overline{C}(d))}{(1-S')kf_i}$ and $S' = \frac{S}{(1-S)f_a+S}$.

Remarks: It shows that the cleaning cost of GREEDY is related to the clustering and skewness within the workload, which can be reflected by the parameters f_a, r_i's and f_i's. In particular, we have the following observations from Equation (6). First, the cleaning cost increases as the active region size increases, since Equation (6) is a monotone increasing function with respect to f_a. Formally, if we take $\overline{C}(d)$ as a function of f_a and take the derivative with respect to f_a on both sides of Equation (6), then we can show that the derivative of $\overline{C}(d)$ is greater than zero. Second, the cleaning cost increases if the workload possesses skewness. To check this observation, we fix $r_i = 1/n$, and study different distributions of f_i's, where $f_1 \leq f_2 \leq \cdots \leq f_n$. If we define $g(f_i) = (k - \overline{C}(d))r_i/(e^{\frac{(k-\overline{C}(d))r_i}{(1-S')kf_i}} - 1)$, then we can see that $g(f_i)$ is a convex function with respect to f_i. Thus, we have $\sum_{i=1}^{n} g(f_i) \geq ng((\sum_{i=1}^{n} f_i)/n) = ng(1/n)$. This inequality indicates that if f_i's are not equal, which corresponds to some skewed workload as we fix $r_i = 1/n$, then the cleaning cost increases. In other words, the cleaning cost is the lowest when the workload has no skewness (i.e., when f_i's are equal). ∎

Note that when $r_i = f_i\,(\forall i = 1, 2, \cdots, n)$, the workload changes to uniform access to blocks in the active region. We can derive the cleaning cost of GREEDY as well as GRA with window size $d = o(N)$. In fact, this case was well studied in previous work, e.g., [7], and Theorem 2 derives the same results, which we state in Corollary 1.

COROLLARY 1. *If the pages in the active region are uniformly accessed, as $N \to \infty$, the average cleaning cost of GREEDY (as well as GRA with window size $d = o(N)$) is*

$$\overline{C}(d) = -W_0\Big(-\frac{1}{1-S'}e^{-\frac{1}{1-S'}}\Big) \Big/ \Big[\frac{1}{(1-S')k}\Big], \tag{7}$$

where $S' = \frac{S}{(1-S)f_a+S}$ and $W_0(\cdot)$ is one branch of Lambert's W function (where $W_0(x) \in [-1, 0]$ for $x \in [-\frac{1}{e}, 0]$).

[1] The equation can be efficiently solved using Newton's method.

3.3 GRA with Window Size $d \geq N_a$

Now we focus on the case when $d \geq N_a$, which includes the scenario of RANDOM where $d = N$.

THEOREM 3. *As $N \to \infty$, the average cleaning cost $\overline{C}(d)$ of GRA with window size d where $d \geq N_a$ is*

$$\overline{C}(d) = (1 - NS/d)k. \tag{8}$$

Remarks: Theorem 3 shows that the average cleaning cost of RANDOM is independent of both the clustering and skewness within the workload. The reason is that $\overline{C}_a(d)$ remains unchanged regardless of parameters f_a, $\mathbf{r}$ and $\mathbf{f}$. ∎

3.4 GRA with Window Size $d = \alpha N_a$

We now consider values of d which are smaller than the number of active blocks N_a, but can scale with N_a. We let $d = \alpha N_a$ (where α is a constant with $0 < \alpha < 1$), which means that the window size d is comparable to the number of active blocks. In this case, the average cleaning cost of GRA can be derived as follows.

THEOREM 4. *As $N \to \infty$, the average cleaning cost $\overline{C}(d)$ of GRA with window size $d = \alpha N_a$ is the unique solution of Equation (9) in the range $[0, k]$*[1].

$$\overline{C}(d) = \sum_{i=1}^{n} \big[(k - \overline{C}(d))r_i\big] \,/\, \Big[(1+\alpha A_i)e^{(1-\alpha)A_i} - 1\Big], \tag{9}$$

where $A_i = \frac{r_i(k-\overline{C}(d))}{(1-S')kf_i}$ and $S' = \frac{S}{(1-S)f_a+S}$.

Remarks: Note that Equation (9) is also a monotone increasing function of f_a. If we fix $r_i = 1/n$, then each summation term in the right hand side of Equation (9) is also a convex function of f_i. Therefore, Equation (9) has the same implications as those in Equation (6). That is, the relationship between the cleaning cost and data locality is the same for GRA with window size $d < N_a$. ∎

4. LOCALITY-AWARE GC

In this section, we consider the case of locality-aware GC, and analyze a design strategy leveraging on data locality. Since it is a general consensus that cleaning cost of GC algorithms may be reduced when data can be differentiated [11, 14] and a workload includes $n + 1$ types of data pages, it is of interest to study the performance of GC that exploits data locality using *data grouping*, which differentiates different types of data pages and stores them separately in different regions in an SSD. In particular, we analyze how data grouping influences the performance of GC in SSD, and how much is the influence for workloads with different degrees of locality? We assume that we have the complete priori information about the workload so as to take full advantage of hot/cold information.

We first illustrate the architecture of an SSD with data grouping. To perform locality-aware GC, we divide the whole storage space into $n + 1$ regions, and let the last region keep the inactive data so that this region never gets

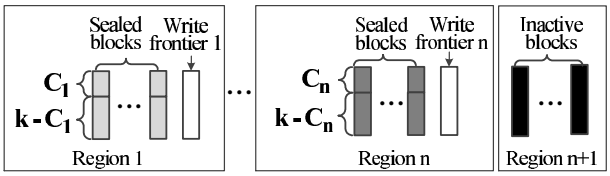


Figure 4: System architecture with data grouping.

accessed. For other n regions, each region keeps only one type of data pages and performs GC independently, so it has a separate write frontier for handling writes directed to that region. Without loss of generality, we assume that region 1 stores the hottest data while region n stores the coldest data. Note that logical pages in each of the first n regions are randomly and uniformly accessed because of data grouping. Hence, the first n regions can be considered as n independent sub-systems, each of which is fed with a uniform workload. The architecture of an SSD with data grouping can be further illustrated in Figure 4. Here, since each region is fed with a uniform workload, it only contains one type of data. In particular, we denote C_i as the average cleaning cost in region i ($1 \leq i \leq n$).

Note that the purpose of considering data grouping is to leverage data locality in improving the GC performance, and the analysis in §3 shows that GREEDY outperforms GRA with a general window size under the uniform workload (i.e., when $f_a = 1$ and $n = 1$). Therefore, in the interest of space, we assume that GREEDY is used for GC in each region for locality-aware GC.

Since there are NS spare blocks in total and $n+1$ separate regions with data grouping, one implementation issue of locality-aware GC is how many spare blocks should be allocated to each region. The allocation of spare blocks to each region may influence the cleaning cost. To study the impact of different allocations, we analyze the cleaning cost of GC with a given allocation $\mathbf{b} = (b_1, b_2, \cdots, b_n)$, where b_i denotes the proportion of spare blocks allocated to region i. We have $0 \leq b_i \leq 1$ and $\sum_{i=1}^{n} b_i = 1$. Note that since no request goes to region $n+1$, we do not need to allocate spare blocks to it. Given the allocation of $\mathbf{b}$, the total number of physical blocks in region i is $N(1-S)f_a f_i + NSb_i$ ($1 \leq i \leq n$), which we denote as N_i. Now we can derive the cleaning cost of locality-aware GC, and the result is stated in Theorem 5.

THEOREM 5. *Given the spare block allocation* $\mathbf{b}$ *where proportion* b_i *of spare blocks are allocated to region* i *(*$1 \leq i \leq n$*) and* $\sum_{i=1}^{n} b_i = 1$ *(*$0 \leq b_i \leq 1$*), the cleaning cost* $\mathcal{C}$ *of locality-aware GC are*

$$\mathcal{C} \;=\; \sum_{i=1}^{n} \big[Lr_i/(k-C_i)\big]C_i,$$

where $C_i = -W_0\big(-\frac{1}{1-S_i}e^{-\frac{1}{1-S_i}}\big)/\big[\frac{1}{(1-S_i)k}\big]$ *as* $N \to \infty$ *and* $S_i = Sb_i/[(1-S)f_a f_i + Sb_i]$.

Remarks: Theorem 5 shows that the allocation of spare blocks, $\mathbf{b}$, affects the cleaning cost of locality-aware GC. The most significant allocation is the one that minimizes the cleaning cost, which we denote as $\mathbf{b}_c^*$. We will further show the performance improvement of cleaning cost for $\mathbf{b}_c^*$ in §5.3 via numerical analysis.

Note that the design strategy of data grouping exploits data locality by eliminating the skewness within a workload. In particular, it separates a highly-skewed workload into multiple uniform workloads with no skewness so as to reduce the cleaning cost (see §5.2 for the impact of skewness). ■

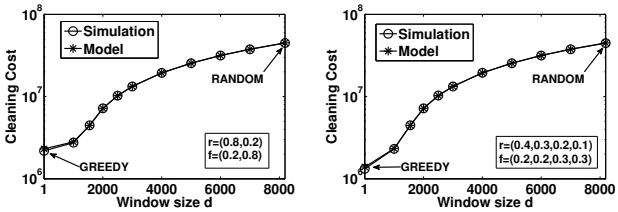
5. MODEL VALIDATION AND IMPLICATIONS

In this section, we conduct model analysis to study the impact of data locality and the implications of locality awareness on the GC design. We first validate our model using the SSD simulator [1]. Based on our model, we then study the impact of data locality and architectural design.

5.1 Model Validation

Since the analysis of locality-aware GC builds on locality-oblivious GC and each region of the SSD in the locality-aware GC corresponds exactly to the case of locality-oblivious GC under uniform workload, our model validation focuses on locality-oblivious GC only with respect to different workloads. To speed up our simulation, we consider a small-scale SSD that contains one flash chip composed of 8 planes with 1024 blocks each. We configure each block with 64 pages of size 4KB each. We use the default timing parameters set by the simulator. We set the spare factor $S = 0.1$ (i.e., the logical address space is 90% of the physical address space).

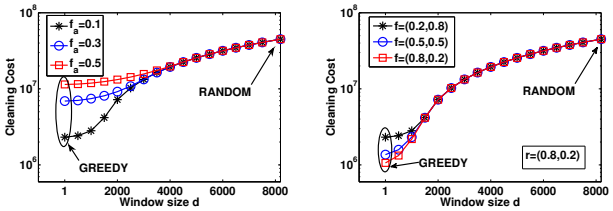
We generate multiple synthetic workloads to drive our simulation. Since read requests do not affect GC, we generate write requests only. Specifically, for each workload, we generate 5M write requests, which suffice to make all active pages get enough number of accesses so to guarantee the statistical significance. Each request has size of 4KB, which aligns with the page size. We specify a workload based on the parameterizing (f_a, n, $\mathbf{r}$, $\mathbf{f}$). Here, we consider two types of workloads: (a) *skewed workload*, in which we have 20% of active pages accessed by 80% of requests by setting $f_a = 0.1$, $n = 2$, $\mathbf{r} = (0.8, 0.2)$ and $\mathbf{f} = (0.2, 0.8)$; and (b) *fine-grained workload*, in which we set a larger value of n by fixing $f_a = 0.1$, $n = 4$, $\mathbf{r} = (0.4, 0.3, 0.2, 0.1)$ and $\mathbf{f} = (0.2, 0.2, 0.3, 0.3)$.



(a) Skewed workload (b) Fine-grained workload

Figure 5: Model validation for locality-oblivious GC.

Figure 5 shows both the model and simulation results based on the above workloads. Each data point in a curve corresponds to GRA with a particular value of d, which increases from the leftmost point (i.e., GREEDY) to the rightmost point (i.e., RANDOM). Our model accurately characterizes the cleaning cost under different workloads. In particular, comparing the model and simulation results for a given value of d, the cleaning costs differ by no more than 7%, and it is no more than 1% in most cases. We also validate other types of workloads, such as different values of f_a.



(a) Impact of clustering (b) Impact of skewness

Figure 6: Impact of data locality on locality-oblivious GC.

The model and simulation results conform to each other. In the interest of space, we do not present the results here.

5.2 How Data Locality Affects Locality-oblivious GC?

We use our model to obtain numerical results, and study how data locality affects the GC performance. Here, we focus on locality-oblivious GC. To study the impact of data locality, we consider a given GC implementation by fixing the window size d of GRA, and examine its changes of cleaning cost under different workloads.

Figure 6(a) shows the impact of clustering by varying $f_a = 0.1$, 0.3, and 0.5, which determines the proportion of the active region, while fixing the skewness with $n = 2$, $\mathbf{r} = (0.8, 0.2)$, and $\mathbf{f} = (0.2, 0.8)$. Figure 6(b) studies the impact of skewness, and we fix $\mathbf{r} = (0.8, 0.2)$, and vary $\mathbf{f} = (0.2, 0.8)$ (which is the most skewed workload), $\mathbf{f} = (0.5, 0.5)$, and $\mathbf{f} = (0.8, 0.2)$ (which is equivalent to the non-skewed workload). From the figures, we see that cleaning cost increases as either the clustering or skewness increases. Moreover, the increase is more pronounced for a smaller d, and GREEDY shows the most increase. For example, in Figure 6(b), the cleaning cost of GREEDY increases from 1.063×10^6 to 2.314×10^6 when we increase skewness by varying from $\mathbf{f} = (0.8, 0.2)$ to $\mathbf{f} = (0.2, 0.8)$. As d increases, both the impact of clustering and skewness on the cleaning cost vanishes. In particular, data locality within a workload, including both clustering and skewness, has a significant impact on GREEDY, but has no impact on RANDOM.

5.3 Will Locality Awareness Help?

We now analyze via our model the performance of locality-aware GC with data grouping (see §4), and study if it improves over locality-oblivious GC.

Figure 7(a) shows the impact of different spare block allocations on GC performance. Here we consider the skewed workload with $f_a = 0.1$, $n = 2$, $\mathbf{r} = (0.8, 0.2)$ and $\mathbf{f} = (0.2, 0.8)$. Each data point (asterisk) corresponds to a particular allocation $\mathbf{b} = (b_1, 1 - b_1)$, where the leftmost data point ($\mathbf{b}_c^* = (0.432, 0.568)$) minimizes the cleaning cost, and the rightmost data point ($\mathbf{b}_w^* = (0.862, 0.138)$) shows the same performance of locality-oblivious GC with GREEDY, which is marked as square. That is, there is no performance gain for locality-aware GC under the block allocation $\mathbf{b}_w^*$. So we focus on the range of allocations from $\mathbf{b}_c^*$ to $\mathbf{b}_w^*$, and other data points are obtained by varying b_1 from 0.432 to 0.862 with a step size of 0.02. The figure shows that data grouping may efficiently reduce the cleaning cost. For example, it can be reduced from 2.31×10^6 to 0.53×10^6 with a proper configuration of the spare blocks in each region.

Figures 7(b) and 7(c) show the impact of clustering and skewness on locality-aware GC, respectively. In Figure 7(b), we vary $f_a = 0.1$, 0.3, and 0.5, while fix the skewness within the workload with $n = 2$, $\mathbf{r} = (0.8, 0.2)$ and $\mathbf{f} = (0.2, 0.8)$. In Figure 7(c), we fix $f_a = 0.1$, $\mathbf{r} = (0.8, 0.2)$, and vary from $\mathbf{f} = (0.2, 0.8)$ to $\mathbf{f} = (0.5, 0.5)$. For each workload, we consider the block allocations varying from $\mathbf{b}_c^*$ to $\mathbf{b}_w^*$ as in Figure 7(a). From Figure 7(b), we see that when the active region size increases, the relative change of cleaning cost decreases. For instance, when $f_a = 0.1$, the cleaning cost under the allocation of $\mathbf{b}_w^*$ (which is 2.31×10^6) is 4.36 times higher than that under the allocation of $\mathbf{b}_c^*$ (which is 0.53×10^6), while this ratio decreases to 2 when $f_a = 0.5$. However, Figure 7(c) shows that the relative change of cleaning cost increases if the workload is more skewed. This can be observed from that the difference of the cleaning cost between the allocations of $\mathbf{b}_c^*$ and $\mathbf{b}_w^*$ is larger when $\mathbf{f} = (0.2, 0.8)$. Even though skewness worsens the cleaning performance of GREEDY for locality-oblivious GC (see Figure 6(b)), it improves the performance of locality-aware GC. That is, incorporating locality awareness into GC design is more significant for workload with higher skewness.

5.4 Summary of Findings

We summarize the implications from our model analysis.

- Data locality affects the performance of locality-oblivious GC with a given implementation (i.e., GRA with a fixed d): (1) cleaning cost increases when either the active region size or the skewness increases, (2) the increase is more pronounced for a smaller window size. In particular, data locality has the most significant impact on GREEDY, but has no impact on RANDOM.
- Locality-aware GC reduces the cleaning cost through data grouping, while the performance improvement depends on the proper allocation of spare blocks. Data locality also affects the performance of locality-aware GC in different ways: (1) if the active region size increases, the relative change of cleaning cost decreases; (2) if the workload has higher skewness, the relative change increases, and so locality awareness is more important if the workload is more skewed.

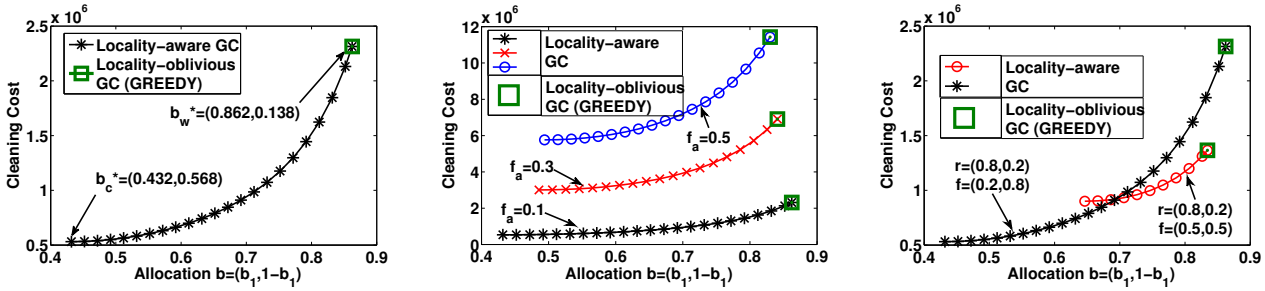
6. TRACE-DRIVEN EVALUATION

In this section, we conduct trace-driven evaluation on the GC performance under real-world workloads, using the SSD simulator [1]. The goals of our evaluation are two-fold. First, we further validate the implications of our model under real-world workload traces. Second, we show the significance of locality awareness on the GC design.

6.1 Evaluation Setup

We have collected eight real-world workload traces to drive our evaluation. Since read requests do not affect the GC performance, we focus on write-intensive traces. The traces can be categorized into three types of environments:

- Financial [25]: It is an I/O trace of an online transaction process application. We select the write-intensive one Financial1.spc in the repository.
- Webmail, Online, Webmail+Online [28]: The three traces describe workloads in a university department. Webmail describes the workload of a mail server, Online



(a) Impact of locality awareness (b) Impact of clustering (c) Impact of skewness

Figure 7: Performance of locality-aware GC.

Traces	Total # of single-page writes	Volume size	Proportion of accessed pages
Financial	4.9 M	4 GB	12.4%
Webmail	6.4 M	18 GB	4.6%
Online	4.2 M	8 GB	3.3%
Webmail+Online	10.6 M	18 GB	5.2%
stg_0	5.2 M	2 GB	3%
prxy_0	23.8 M	2 GB	2%
proj_0	41.0 M	2 GB	3.2%
prn_0	18.1 M	2 GB	2.4%

Table 1: Statistics of traces.

describes a coursework management workload on Moodle, and Webmail+Online is the combination of Webmail and Online.

- prxy_0, prn_0, proj_0, stg_0 [20]: The four traces are collected from an enterprise data center. The original set of traces cover 36 volumes in total. We choose four of them that have high write-to-read ratios and a sufficiently large number of write requests.

Table 1 summarizes the statistics of traces. We fix the page size as 4KB and align the requests in each trace to be a multiple of the page size. As shown in the table, each trace consists of millions of single-page writes. We then determine the volume size (i.e., the capacity of the logical space) of each trace by multiplying the largest logical page number among all requests with the page size and rounding it up to the next multiple of gigabytes. For the Financial trace, we ignore the requests with application-specific unit numbers ASU1, ASU3, and ASU5 as they lead to an extremely large volume size. As shown in Table 1 Webmail and Webmail+Online have a large volume size, while the data-center traces [20] have a much smaller volume size. By measuring the proportion of logical pages that are accessed by at least one write request, we see that all write requests only access a very small portion of the logical space for all traces (i.e., f_a is small), with up to only 12.4% as shown in Table 1. We use this characteristic to motivate our workload model formulation (see §2.2).

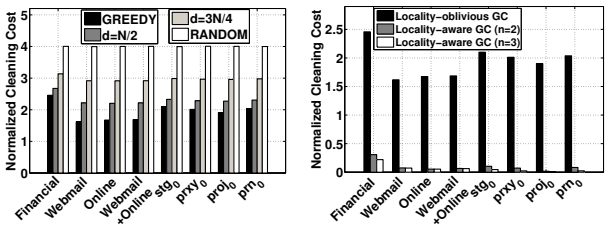
We configure the SSD simulator for our evaluation as follows. We note that each flash chip in an SSD typically operates independently with its own I/O bus, and performs GC on its own blocks independently of other flash chips. Thus, when we evaluate a GC algorithm under a given workload, we feed the trace to an SSD with only one flash chip so as to preserve the statistics of the workload. We set each block with the number of pages $k = 64$, and configure the SSD capacity (i.e., the total number of blocks) according to the volume size of each trace, and the spare factor which is set as $S = 0.2$. We focus on page-level address mapping, and assume that the SSD is fully utilized initially, and locality pages are mapped sequentially to physical pages. Thus, each write request to a page corresponds to an update. The full initial state is the default setting of the SSD simulator, and we use it to stress-test the I/O performance of the SSD. Note that the SSD contains a proportion S of spare blocks that have no data initially. To eliminate the influence of these spare blocks, we first warm up the simulator with 10M write requests that uniformly access the pages of the entire logical address space. Afterwards, we drive our evaluation with the real-world traces.

6.2 Impact of Data Locality on Cleaning Cost

Locality-oblivious GC: We first evaluate the cleaning cost of locality-oblivious GC. We consider different GC implementations by configuring GRA with $d = 1$, $\frac{N}{2}$, $\frac{3N}{4}$, and N, where N is the total number of physical blocks. Note that $d = 1$ and $d = N$ correspond to GREEDY and RANDOM, respectively. Since the workload size varies from traces, for ease of comparison, we normalize the cleaning cost as the number of internal page writes caused by GC over the number of external page writes containing in the workload. This normalized cleaning cost can be interpreted as write amplification [12] minus one.

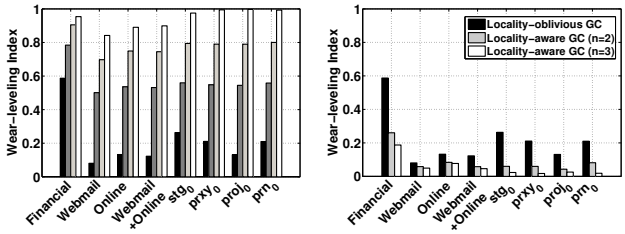
Figure 8(a) shows the results. We see that the performance of locality-oblivious GC manifests in two ways. First, RANDOM gives the worst performance in terms of cleaning cost and GREEDY achieves the best performance. In fact, a trade-off between performance and durability exists for GRA with different values of d, and we show the trade-off in §6.3. Second, we observe that among different GC implementations, GREEDY has the most varying cleaning cost across different workloads. Its normalized cleaning cost varies from 1.6 to 2.46. RANDOM has the least variance in cleaning cost, as also indicated by our analysis (see §3).

Locality-aware GC: Now we evaluate the cleaning cost of locality-aware GC. Our implementation assumes the complete knowledge of the workload, so that we can evaluate the best achievable design of a GC algorithm. Specifically, we fix the number of regions n, and then measure the access frequency of each data page so as to separate data pages into n regions and obtain the corresponding parameters $\mathbf{r}$ and $\mathbf{f}$, as defined in §2.2. In terms of the spare block allocation for each region, since different allocations lead to different per-



(a) Locality-oblivious GC (b) Locality-aware GC

Figure 8: Cleaning cost of GC algorithms.



(a) Locality-oblivious GC (b) Locality-aware GC

Figure 9: Wear-leveling index of GC algorithms.

formance results (see §4), we focus on the one minimizing the cleaning cost.

We consider two scenarios of data grouping: (1) we separate data pages into $n = 2$ types, one of which has data pages accessed by at least 10 times and another has data pages accessed by less than 10 times; (2) we separate data pages into $n = 3$ types, which have data pages accessed by no less than 50 times, less than 50 but no less than 10 times, and less than 10 times, respectively. For comparison, we also plot the cleaning cost of GREEDY in locality-oblivious GC.

Figure 8(b) shows the cleaning cost of locality-aware GC for different traces. First, compared to locality-oblivious GC, cleaning cost can be *significantly reduced with data grouping*, e.g., by more than 87% for locality-aware GC with $n = 2$. The reduction can even reach up to 99% for the proj_0 trace. Note that we can achieve further reduction, although marginal, if we separate data pages into three types.

6.3 Impact of Data Locality on GC Design Trade-off

As stated in [1, 17], GC design poses a trade-off between performance and durability of an SSD. In this subsection, we show the performance-durability trade-off of GC design and study the impact of data locality on it.

Recall that we use the cleaning cost as defined in Equation (1) to represent the performance of GC algorithms. For durability, we define another metric, namely *wear-leveling index* $\mathcal{W}(alg)$.

$$\mathcal{W}(alg) = \left[N\sum_{i=1}^{N}\left(\frac{n_i}{\sum_{i=1}^{N} n_i}\right)^2\right]^{-1} = \left(\sum_{i=1}^{N} n_i\right)^2 / \left(N\sum_{i=1}^{N} n_i^2\right),$$

where n_i $(1 \le i \le N)$ is the number of erasures performed on block i. The intuition of $\mathcal{W}(alg)$ is that it quantifies the fairness among n_i's, so it characterizes how *evenly* blocks are erased throughout all GC operations. Note that we have $0 < \mathcal{W}(alg) \le 1$, where $\mathcal{W}(alg) = 1$ means all blocks receive the same number of erasures.

Figure 9 shows the wear-leveling index of different GC algorithms for both locality-oblivious GC and locality-aware GC. By combining the results in Figure 8, we see that the design trade-off manifests in two ways for locality-oblivious GC. First, the trade-off exists in GRA with different values of d. For a given trace, RANDOM gives a large wear-leveling index but introduces high cleaning cost, while GREEDY incurs low cleaning cost but has a small wear-leveling index. Second, the trade-off exists among different traces. For instance, GREEDY shows the highest cleaning cost and wear-leveling index under the Financial trace, which spans the largest active region (i.e., f_a is the largest) as indicated in Table 1. From our analysis, a larger f_a implies higher cleaning cost (see Figure 6). For locality-aware GC, when cleaning cost decreases, the wear-leveling index also decreases (by 28% to 77% as shown in Figure 9(b)). It is thus important to deploy a dedicated wear-leveling technique orthogonal to locality-aware GC.

6.4 Summary

The above evaluation results show that the impact of data locality varies across different GC algorithms. For locality-oblivious GC, data locality significantly influences the performance of GREEDY, but has negligible impact on RANDOM. We demonstrate that locality-aware GC can efficiently reduce the cleaning cost. We also show that the GC design poses a trade-off between cleaning and wear leveling, and the design trade-off exists under all workloads.

7. RELATED WORK

NAND-flash-based SSDs have received much attention in both industrial and academic communities. Several aspects are studied such as write performance optimization [2,10,24], reliability analysis [16], and lifetime extension [5,18]. In particular, Gal *et al.* [8] survey the algorithms and data structures of flash memory. Chen *et al.* [4] and Jung *et al.* [13] reveal the intrinsic characteristics and system implications of SSDs through extensive empirical measurements. Agrawal *et al.* [1] study different design trade-off issues for SSDs using a trace-driven simulator.

Since the cleaning operation introduces additional writes (also known as write amplification) that are critical to SSD performance, several studies analyze the cleaning performance of GC algorithms. For example, Hu *et al.* [12] propose a probabilistic model to analyze a windowed greedy algorithm that erases the block with the fewest valid pages among the least-recently-used blocks. Bux *et al.* [3] analyze the greedy algorithm under uniform workload. Desnoyers [7] analyzes the greedy algorithm under both uniform and non-uniform workloads. Van Houdt [26] develops a mean field model to derive the write amplification for various GC algorithms under uniform workload, and later extends the model for hot/cold workload [27]. Yang and Zhu [29] further employ the mean field model to analyze the performance of various hotness-aware GC algorithms. Li *et al.* [17] also propose a mean field model to analyze different GC algorithms for SSDs, with the emphasis on modeling the trade-off between cleaning and wear leveling.

Unlike most previous studies, our work focuses on studying the impact of data locality on the cleaning performance of different GC implementations, and makes the following differences. First, we consider more general workloads, with different levels of hotness/coldness, by modeling both clustering and fine-grained skewness. Second, we analyze a family of locality-oblivious GC algorithms and consider a more general implementation of locality-aware GC, which supports the separation of data pages in more fine-grained levels. Finally, we conduct trace-driven simulations to validate our analysis and the findings of our model. In summary, our work complements the previous studies by addressing more general deployment scenarios through both analytical modeling and trace-driven simulations.

8. CONCLUSIONS

We develop analytical models to characterize the impact of data locality on the GC performance of SSDs. We first formalize a workload model to capture data locality (i.e., clustering and skewness) evidenced by the trace analysis of real-life I/O workloads. We then integrate the workload model into GC performance analysis, and study two classes of GC designs: locality-oblivious GC and locality-aware GC. In locality-oblivious GC, we analyze the cleaning cost of a family of GC algorithms. We show that as the active region size increases, the cleaning cost increases, and a more skewed workload also increases the cleaning cost. Furthermore, we extend our analysis for locality-aware GC. We show that data grouping can reduce the cleaning cost, and the reduction is more significant if the workload is more skewed, which shows the importance of locality awareness. Finally, we conduct extensive trace-driven simulations to validate the accuracy and findings of our models. The main focus of our work is to analytically study how data locality influences the cleaning performance of various GC algorithms so as to guide the GC design, while we leave the actual design of locality-aware GC algorithms that exploit workload locality and adapt to workload dynamics as future work.

9. ACKNOWLEDGMENTS

The work of Yongkun Li was supported in part by National Nature Science Foundation of China under Grant No. 61303048, and the Fundamental Research Funds for the Central Universities under Grant No. WK0110000040.

10. REFERENCES

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design Tradeoffs for SSD Performance. In *Proc. of USENIX ATC*, Jun 2008.

[2] A. Birrell, M. Isard, C. Thacker, and T. Wobber. A Design for High-performance Flash Disks. *ACM SIGOPS Oper. Syst. Rev.*, 41(2):88–93, Apr 2007.

[3] W. Bux and I. Iliadis. Performance of Greedy Garbage Collection in Flash-based Solid-state Drives. *Performance Evaluation*, Nov 2010.

[4] F. Chen, D. A. Koufaty, and X. Zhang. Understanding Intrinsic Characteristics and System Implications of Flash Memory based Solid State Drives. In *Proc. of ACM SIGMETRICS*, Jun 2009.

[5] F. Chen, T. Luo, and X. Zhang. CAFTL: A Content-aware Flash Translation Layer Enhancing the Lifespan of Flash Memory Based Solid State Drives. In *Proceedings of USENIX*, FAST, 2011.

[6] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song. System Software For Flash Memory: A Survey. In *Proc. of Int. Conf. on Embedded and Ubiquitous Computing*, Aug 2006.

[7] P. Desnoyers. Analytic Modeling of SSD Write Performance. In *Proceedings of SYSTOR*, Jun 2012.

[8] E. Gal and S. Toledo. Algorithms and Data Structures for Flash Memories. *ACM Computing Surveys*, 37(2):138–163, Jun 2005.

[9] A. Gupta, Y. Kim, and B. Urgaonkar. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. In *Proc. of ACM ASPLOS*, Mar 2009.

[10] A. Gupta, R. Pisolkar, B. Urgaonkar, and A. Sivasubramaniam. Leveraging Value Locality in Optimizing NAND Flash-based SSDs. In *Proc. of USENIX FAST*, 2011.

[11] J.-W. Hsieh, T.-W. Kuo, and L.-P. Chang. Efficient Identification of Hot Data for Flash Memory Storage Systems. *ACM TOS*, Feb 2006.

[12] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. Write Amplification Analysis in Flash-based Solid State Drives. In *Proc. of SYSTOR*, May 2009.

[13] M. Jung and M. Kandemir. Revisiting Widely Held SSD Expectations and Rethinking System-level Implications. In *Proc. of ACM SIGMETRICS*, Jun 2013.

[14] H.-S. Lee, H.-S. Yun, and D.-H. Lee. HFTL: Hybrid Flash Translation Layer based on Hot Data Identification for Flash Memory. *IEEE Trans. on Consumer Electronics*, 55(4):2005–2011, 2009.

[15] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song. A Log Buffer-based Flash Translation Layer Using Fully-associative Sector Translation. *ACM TECS*, 6(3), Jul 2007.

[16] Y. Li, P. P. C. Lee, and J. C. S. Lui. Stochastic Analysis on RAID Reliability for Solid-State Drives. In *Proc. of IEEE SRDS*, 2013.

[17] Y. Li, P. P. C. Lee, and J. C. S. Lui. Stochastic Modeling of Large-Scale Solid-State Storage Systems: Analysis, Design Tradeoffs and Optimization. In *Proc. of ACM SIGMETRICS*, 2013.

[18] Y. Lu, J. Shu, and W. Zheng. Extending the Lifetime of Flash-based Storage through Reducing Write Amplification from File Systems. In *Proc. of USENIX FAST*, 2013.

[19] Micron Technology. Bad Block Management in NAND Flash Memory. Technical Note, TN-29-59, 2011.

[20] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. *ACM TOS*, 4(3):10:1–10:23, Nov 2008.

[21] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim. A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND Flash-based Applications. *ACM TECS*, 7(4):38:1–38:23, Aug 2008.

[22] Z. Qin, Y. Wang, D. Liu, and Z. Shao. Demand-based Block-level Address Mapping in Large-scale NAND Flash Storage Systems. In *Proc. of IEEE/ACM/IFIP CODES+ISSS*, Oct 2010.

[23] M. Rosenblum and J. K. Ousterhout. The Design and Implementation of a Log-structured File System. *ACM Trans. Comput. Syst.*, 10(1):26–52, Feb 1992.

[24] A. Soga, C. Sun, and K. Takeuchi. NAND Flash Aware Data Management System for High-speed SSDs by Garbage Collection Overhead Suppression. In *IEEE 6th International Memory Workshop (IMW)*, May 2014.

[25] Storage Performance Council. `http://traces.cs.umass.edu/index.php/Storage/Storage`, 2002.

[26] B. Van Houdt. A Mean Field Model for a Class of Garbage Collection Algorithms in Flash-based Solid State Drives. In *Proc. of ACM SIGMETRICS*, Jun 2013.

[27] B. Van Houdt. Performance of Garbage Collection Algorithms for Flash-based Solid State Drives with

Hot/cold Data. *Performance Evaluation*, 70(10):692 – 703, Sep 2013.

[28] A. Verma, R. Koller, L. Useche, and R. Rangaswami. SRCMap: Energy Proportional Storage using Dynamic Consolidation. In *Proc. of USENIX FAST*, Feb 2010.

[29] Y. Yang and J. Zhu. Analytical Modeling of Garbage Collection Algorithms in Hotness-aware Flash-based Solid State Drives. In *Proc. of IEEE MSST*, June 2014.

APPENDIX

Proof of Theorem 1 in §3.1: Since the proportion of active region in logical space is f_a, the number of active blocks is $N_a = N((1-S)f_a+S) - 1$. We consider two cases.
Case 1: $d \leq N_a$. Based on the argument that the blocks that are sealed earlier should contain fewer valid pages on average, we can make an approximation that the d candidate blocks are chosen from the d blocks that are sealed in the earliest time. Thus, only active blocks have the chance of being selected for GC if $d \leq N_a$. Moreover, an active block can be reclaimed at the j-th ($j = N_a - d + 1, N_a - d + 2, \cdots, \infty$) GC instant since it has been sealed, and the corresponding probability is $\frac{1}{d}(1-\frac{1}{d})^{j-(N_a-d)-1}$. Note that $\overline{C}_a(d)$ denotes the average number of valid pages in an active block to be reclaimed under GRA with selection window size d. Among these $\overline{C}_a(d)$ pages, the average number of type$-i$ pages is denoted by $\overline{C}_i(d)$. We have $\overline{C}_a(d) = \sum_{i=1}^{n} \overline{C}_i(d)$. To simplify the presentation, let us drop the notation d when the context is clear. Since each GC needs to write back $\overline{C}_a$ valid pages on average, each clean block can only handle $k - \overline{C}_a$ external page writes. Therefore, a sealed block is reclaimed after handling $(j-1)(k-\overline{C}_a)$ external page writes on average if it is reclaimed by the j-th GC. For each external page write, it updates a type-i page with probability $\frac{r_i}{N(1-S)f_a k f_i} = \frac{r_i}{(N_a+1)(1-S')k f_i}$ where $N_a = N((1-S)f_a + S) - 1$ and $S' = \frac{S}{(1-S)f_a+S}$. The physical meaning of S' is that it represents the spare factor of a sub-system without inactive blocks. Now $\overline{C}_i$ can be characterized by the following equation.

$$\begin{aligned}\overline{C}_i &= (\overline{C}_i + (k-\overline{C}_a)r_i) \sum_{j=N_a-d+1}^{\infty} \Big[\frac{1}{d}\Big(1-\frac{1}{d}\Big)^{j-(N_a-d)-1} \times \\ &\quad \Big(1 - \frac{r_i}{(N_a+1)(1-S')kf_i}\Big)^{(j-1)(k-\overline{C}_a)}\Big] \\ &= (\overline{C}_i + (k-\overline{C}_a)r_i)(1-P')^{N_a-d}\Big[\frac{1}{(1+P'\times d)}\Big], (\text{as } N\to\infty)\end{aligned}$$

where $P' = \frac{r_i(k-\overline{C}_a)}{(N_a+1)(1-S')kf_i}$. Thus, $\overline{C}_i$ is derived as in Eq. (5). Now the average cleaning cost incurred by reclaiming an active block is $\overline{C}_a = \sum_{i=1}^{n} \overline{C}_i$. Since inactive blocks will never be chosen for GC as $d \leq N_a$, we have $\overline{C} = \overline{C}_a$.
Case 2: $d > N_a$. There are N_a active blocks and $d - N_a$ inactive blocks in the window of d candidate blocks, and each of them is chosen for GC with equal probability, so the probabilities of reclaiming an active and an inactive block in each GC are N_a/d and $(d-N_a)/d$, respectively. Note that reclaiming an inactive block always incurs k page writes. Thus, the average cleaning cost in each GC can be derived as $(N_a/d)\overline{C}_a(d) + [(d-N_a)/d]k$. Note that all active blocks are in the window and uniformly selected for GC, which is equivalent to the case where $d = N_a$, so $\overline{C}_a(d)$ can also be derived via Equations (4)-(5) by replacing d with N_a.

Given the average cleaning cost $\overline{C}(d)$, the write frontier can only handle $k - \overline{C}(d)$ external page writes during one GC. Therefore, the total cleaning cost caused by GC is $[L/(k - \overline{C}(d))]\overline{C}(d)$.

Proof of Theorem 2 in §3.2: Since $d = o(N)$, according to Equation (5), as $N \to \infty$, $\overline{C}_i(d)$ can be transformed to $\overline{C}_i(d) = [(k-\overline{C}_a(d))r_i]/(e^{A_i}-1)$, where $A_i = r_i(k-\overline{C}(d))/[(1-S')kf_i]$ and $S' = S/((1-S)f_a + S)$. Therefore, the average cleaning cost in each GC, $\overline{C}(d)$, can be easily derived as

$$\overline{C}(d) = \sum_{i=1}^{n} (k - \overline{C}(d))r_i/(e^{A_i}-1). \tag{10}$$

To show the uniqueness of the solution in Equation (10), we only focus on $\overline{C}(d) \in [0,k]$. We let $f(x) = \sum_{i=1}^{n}(k-x)r_i/(e^{A_i(x)}-1) - x$, where $A_i(x) = r_i(k-x)/[(1-S')kf_i]$ and $x \in [0,k]$. We have $f(0) > 0$, and $\lim_{x\to k^-} f(x) < 0$. By checking the first-order derivative of $f(x)$, we have $f'(x) < 0$. So Equation (10) has a unique solution in $[0,k]$.

Proof of Corollary 1 in §3.2: Since $r_i = f_i$ $(i = 1, 2, \cdots, n)$, Equation (6) can be rewritten as $\overline{C}(d) = ke^{-[k-\overline{C}(d)]/[(1-S')k]}$, which can be solved in terms of Lambert's W function as $\overline{C}(d) = -W\big(-\frac{1}{1-S'}e^{-\frac{1}{1-S'}}\big)/\big[\frac{1}{(1-S')k}\big]$. Note that $S' < 1$, we have $-\frac{1}{1-S'}e^{-\frac{1}{1-S'}} \geq -1/e$, so Lambert's W function $W(\cdot)$ is a real valued function and has two branches. One branch defines a single-valued function $W_0(\cdot) \geq -1$, while the other branch has $W_1(\cdot) \leq -1$. In our case, since $\overline{C}(d) \leq k$, we restrict Lambert's W function to the higher branch $W_0(\cdot)$, and we obtain the results as claimed.

Proof of Theorem 3 in §3.3: As $N \to \infty$, we have $N_a \to \infty$, since $d \geq N_a$, based on Eq. (5), we have

$$\overline{C}_i(d) = r_i(k-\overline{C}_a(N_a))/\Big[\frac{r_i(k-\overline{C}_a(N_a))}{(N_a+1)(1-S')kf_i}N_a\Big] = (1-S')kf_i.$$

Therefore, $\overline{C}_a(d) = (1-S')k$ and $\overline{C}(d)$ can be derived as $\overline{C}(d) = (N_a/d)(1-S')k + (1-N_a/d)k = (1-NS/d)k$.

Proof of Theorem 4 in §3.4: Since $d = \alpha N$, as $N \to \infty$, Equation (5) can be simplified as $\overline{C}_i(d) = [(k-\overline{C}_a(d))r_i]/[(1+\alpha A_i)e^{(1-\alpha)A_i} - 1]$, where $A_i = r_i(k-\overline{C}_a(d))/[(1-S')kf_i]$. Now $\overline{C}(d)$ can be easily derived via $\overline{C}(d) = \overline{C}_a(d) = \sum_{i=1}^{n}\overline{C}_i(d)$. For the proof of uniqueness, it is the same as that in Theorem 2.

Proof of Theorem 5 in §4: Since the workload in each region is uniform, based on Corollary 1, we can derive the average cleaning cost in each GC, and we denote it as C_i for region i. We have $C_i = -W_0\big(-\frac{1}{1-S_i}e^{-\frac{1}{1-S_i}}\big)/\big[\frac{1}{(1-S_i)k}\big]$, where $S_i = Sb_i/[(1-S)f_a f_i + Sb_i]$. Note that the workload contains L page writes, and proportion r_i of them go to region i, so the total number of GCs in region i is $\frac{Lr_i}{k-C_i}$ as each GC can only handle $k - C_i$ external page writes. Therefore, the total number of page writes issued by GC in region i is $\frac{Lr_i}{k-C_i}C_i$, and the total cleaning cost is the sum of all page writes issued by GC in all regions. So we have $\mathcal{C} = \sum_{i=1}^{n}\frac{Lr_i}{k-C_i}C_i$.