On the Power of Asymmetry and Memory in Flash-based SSD Garbage Collection

B. Van Houdt

Department of Mathematics and Computer Science
University of Antwerp - iMinds

Abstract

The power of d random choices has been widely recognized in the balls-and-bins, hashing and load balancing setting and various refinements such as the introduction of asymmetry or memory have been proposed. Recently, the d-choices garbage collection algorithm for flash-based SSDs was introduced and shown to provide an excellent tradeoff between performance and simplicity. In this paper we study the impact of asymmetry and memory on the performance of the d-choices garbage collection algorithm using both mean field models and trace-based simulations. Numerical examples demonstrate that both asymmetry and memory reduce the write amplification, however the reduction is typically less than 2% when exploiting asymmetry, while more significant gains exceeding 10% are observed when memory is introduced.

1. Introduction

The power of d random choices, initially studied in a simple balls-and-bins model in [1], has becomes a general and robust paradigm applicable to a wide variety of systems, see [19]. New applications and refinements still appear regularly, some of these refinements include the use of asymmetry [28, 20] and memory [18]. Recently, the power of the d-choices algorithm as a garbage collection (GC) algorithm for flash-based solid state drives (SSDs) was also demonstrated in [25, 14]. In this paper, we introduce the d-left GC algorithm, which exploits asymmetry, and the d-memory GC algorithm, that relies on memory. This paper is an extended version of [24], that was limited to the analysis of the d-memory algorithm. We study the so-called write amplification, which has a profound impact on the write performance and life-span of an SSD, of both these algorithms under synthetic and real life workloads. For the synthetic workloads we consider uniform random writes and extend the mean field model of [25] in two different ways, while for the real life workloads we rely on trace-driven simulation experiments. The main overall finding is that while memory can result in a

Email address: `benny.vanhoudt@uantwerpen.be` (B. Van Houdt) significant reduction of the write amplification, asymmetry seems to offer little benefit.

Data on an SSD is organized in blocks each containing a fixed number of fixed sized pages. The page size is typically a few kilobytes and it is also the unit of data transfer with the SSD. The number of pages per block b is a power of 2 ranging from $b = 32$ to 256 in modern SSDs. The number of physical blocks N on the drive depends on the SSD storage capacity and often exceeds 100,000. At any point in time, a page is either in the valid, invalid or erase state. In order to write data on a page, it must be in the erase state. However, a page cannot be erased individually, instead erase operations must be performed on entire blocks. As such SSDs tend to write new data elsewhere and simply invalidate the data in the old location.

To support out-of-place writes the device maintains a mapping between the logical and physical page numbers, called the flash translation layer (FTL) [5, 7]. In a page-mapped FTL any logical page can be mapped to any physical page at the expense of requiring as many entries in the map as there are pages on the SSD. Many flash-based devices rely on a hybrid-mapped FTL (e.g., [11, 13, 10, 12]) as this reduces the size of the FTL map. However, some of these solutions were designed specifically for mobile embedded systems (e.g., MP3 and PDAs) and are not very suitable for the workloads with random writes and temporal locality characteristics encountered in general-purpose computing.

We focus on a page-mapped FTL and any new data is sequentially written to a special block, called the write frontier (WF). The main task of the garbage collection algorithm is to select a new WF whenever the current one becomes full and ideally it should select blocks with as few valid pages as possible (see Section 2.1 for more details). Recently, the d-choices GC algorithm was shown to provide an excellent tradeoff between simplicity and performance [25]. This GC algorithm was inspired by the well-known d-choices algorithm in load balancing [29, 17], where whenever a job needs to be dispatched d queues are chosen uniformly at random and the job is assigned to the shortest among the d chosen queues. The d-choices GC algorithm behaves similarly as it chooses d blocks uniformly at random and selects the one containing the fewest number of valid pages.

Often several of the d chosen blocks may in fact contain a small number of valid pages and one may wonder whether in general the system performance improves if we keep track of some of the best choices, instead of reselecting d blocks each time the GC algorithm is activated. Adding memory on top of the d choices should be beneficial, but it is less obvious if any gain can be achieved if we keep track of the block ids of the next c best choices and reselect only $d - c$ blocks, for some $c > 0$. The idea of combining memory with d-choices to recall some of the best choices of the previous selection has also been explored in a load balancing setting [18], where the main finding is that memory may increase the mean response time, but it does give rise to improved queue length asymptotics.

Another idea that was used in a load balancing setting to improve the queue length asymptotics is to introduce asymmetry [28, 20]. In this case the N

queues are partitioned into d sets of size N/d, labeled 1 to d, and the d choices are made such that one queue is selected uniformly at random within each set. Further, when ties occur between queues belonging to different sets, they are no longer broken at random. Instead the queue in the set with the smallest label is selected, creating a form of asymmetry in the system. An interesting question is whether this type of asymmetry is also beneficial in a garbage collection setting.

In this paper we study the write amplification of the d-choices GC algorithm with memory, which we call the d-memory GC algorithm, and the d-choices GC algorithm with asymmetry, which we call the d-left GC algorithm. We consider both synthetic and trace-based workloads, where the synthetic workload consists of uniform random writes, meaning all pages are accessed equally often and there is no temporal or spacial locality, and the write amplification is analyzed by extending the mean field model of [25] in two ways. Although this extension is not hard, the resulting set of ODEs that captures the evolution of the mean field model for the d-memory GC algorithm relies on the steady state probabilities of a Markov chain with $(b+1)^c$ states, where c represents the number of block ids stored by the d-memory GC algorithm and b denotes the number of pages per block. Hence, for realistic values of b and c, e.g., $b = 64$ and $c = 10$, this ODE cannot be solved in a direct manner. Instead we show that the necessary steady state probabilities of these $(b+1)^c$-state Markov chains can be expressed through the steady state probabilities of b different Markov chains, each having $c+1$ states only.

For the trace-based workloads, we rely on simulation experiments and analyze the write amplification of SSDs that either make use of a single or the double write frontier as described in [26]. As such the paper makes the following contributions:

1. We extend the mean field model of [25] in two ways to analyze the write amplification of the d-left and d-memory GC algorithm under uniform random writes. For the d-memory GC algorithm we use lumpability properties to solve the resulting ODE in an efficient manner for realistic parameter values. We validate both mean field models by means of simulation experiments and present numerical results to demonstrate the added value of incorporating asymmetry or memory.
2. We use trace-based simulation experiments to confirm that the main findings of the mean field model that relies on uniform random writes, also apply under real workloads. As real workloads contain both hot and cold data we consider SSDs that operate either using a single or a double write frontier (see Section 2.1 for details). In the latter case adding more memory does not always reduce the write amplification, that is, there exists an optimal amount of memory.
3. The main overall finding is that while memory can result in a significant reduction of the write amplification (exceeding 10%), asymmetry seems to offer rather limited benefit.

The paper is structured as follows. Section 1.1 discusses some related work, while Section 2 described the system operation and introduces the d-left and

d-memory GC algorithm. The mean field models are presented and validated in Section 3, while various numerical results are presented in Section 4. In Section 5 we look at the impact of using real-life workloads, while conclusions are drawn in Section 6.

1.1. Related work

The write amplification (and the distribution of the number of valid pages per block) of the following garbage collection algorithms was analyzed in a number of earlier studies:

- The FIFO GC algorithm [23, 30, 6] selects the blocks in a cyclic order.
- The greedy GC algorithm [15, 3, 6] selects the block containing the fewest number of valid pages among all the blocks.
- The d-choices GC algorithm [25, 14, 26] selects the block with the fewest number of valid pages out of a set of d randomly chosen blocks.
- The windowed GC algorithm [9] maintains a window containing the w least recently selected blocks and selects the block with the fewest number of valid pages in the current window.

Most of the above studies consider uniform random writes and focus on the case where the number of blocks N tends to infinity. Under uniform random writes the greedy algorithm is believed to be optimal, while the FIFO algorithm often has the worst write amplification. Both the d-choices and the windowed algorithm provide a trade-off between the simplicity of FIFO and the performance of the greedy algorithm (note, the windowed algorithm corresponds to the FIFO and greedy GC algorithm when the window size equals 1 and N, respectively). The d-choices GC algorithm however provides a much better trade-off than the windowed algorithm as small values of d suffice, e.g., $d = 10$, to achieve a write amplification close to that of the greedy algorithm (see [25] for details).

Far fewer results are available in case of non-uniform random writes, in this case some of the data, termed the hot data, is accessed more often, while the remaining data is termed the cold data. The performance of the FIFO and greedy algorithm in the presence of hot and cold data was analyzed in [6] for a system using a single write frontier. These results indicated that the write amplification worsens significantly as the hot data gets hotter, especially for the FIFO GC algorithm. Similar results were provided in [26] for the d-choices GC algorithm, where additionally the concept of the double write frontier was described. With the double write frontier the write amplification was shown to decrease as the hot data gets hotter and the greedy algorithm was shown to be no longer optimal, instead there exists an optimal value for d. A simple Rosenblum data model for the hot and cold data was used in [6, 26], though some trace-based results for FIFO were also provided in [6]. Under the Rosenblum model a fraction f of the logical address space corresponds to *hot* data and the remaining fraction to *cold* data. The fraction of write operations to the hot data is denoted as r. Note, in the Rosenblum model there is no spacial locality in the workload and the temporal locality does not vary over time.

2. System description and algorithms

2.1. System operation

As indicated in the introduction, we focus on a page-mapped FTL where new data is sequentially written to a special block, called the write frontier (WF). Suppose at some point in time that the first k pages of the WF are in the valid/invalid state, while the remaining $b-k$ pages are in the erase state. The data of the next write operation is stored on page $k+1$ of the WF, while altering its state from erase to valid. The data in the old location is invalidated and the FTL mapping is updated. When the WF becomes full, meaning all its pages are in the valid/invalid state, the GC algorithm creates a new WF. It does this by selecting a block (in some algorithm dependent manner) and by erasing the entire block after copying any valid data that was left on the block to memory. Assume there were j valid pages in the selected block before the erase operation took place. After the erase operation the j valid pages are written back to the selected block using the first j pages and the next $b-j$ externally requested write operations can make use of the last $b-j$ pages of the newly created WF. It should be noted that in practice, the copy to memory is avoided by making use of a *clean* block, but the performance with or without the clean block is identical.

Note, whenever the GC algorithm selects a block containing j valid pages, the SSD needs to perform one erase and b write operations in order to process $b-j$ externally requested write requests (i.e., requests issued by the operating system). The ratio of the total number of writes to the total number of externally requested writes is defined as the write amplification (WA). If we denote p_j as the probability that the GC algorithm selects a block with j valid pages, the WA can be expressed as

$$WA = \frac{b}{b - \sum_{j=0}^{b} j p_j}.$$

The write amplification is an important performance measure because it directly impacts the speed of the drive (erase and write operations are much slower than read operations [4]) as well as the lifespan of the SSD (as each block can only be erased a limited number of times [8]).

The above discussion applies to an SSD that makes use of a single WF. The double WF described in [26] makes use of 2 WFs: the WFI is used for internally requested writes and the WFE is used for externally requested writes. Whenever the GC algorithm is activated it attempts to create a WFE with all of its pages in the erase state. It does this by first selecting a block (in some specific manner), where we denote the number of valid pages in the selected block as j. If the WFI has j or more pages in the erase state, the GC algorithm copies the j valid pages of the selected block to the WFI (changing the state of j of its pages from erase to valid) and erases the selected block, which becomes the new WFE with all of its pages in the erase state. If the WFI does not contain enough pages in the erase state, only some of the valid pages of the selected block are copied to the WFI, while the remaining pages are copied to memory and placed back on the selected block after the erase operation took place. In the latter case the newly selected block becomes the new WFI and the GC algorithm is immediately activated again in order to create a new WFE. For the double WF the write amplification is still defined as the ratio of the total number of writes over the total number of externally requested writes (see [26] for more details). It is worth noting that the idea of separating writes triggered by the operating system and writes triggered by garbage collection has also been proposed in the context of log structured file systems [16, Section 7].

SSDs also rely on over-provisioning, meaning a drive with N physical blocks is perceived by the operating system as having a storage capacity of $U < N$ blocks. The spare factor S_f is defined as $1 - U/N$ and ranges from 0.02 to 0.2 in real life SSDs. The advantage of over-provisioning is that a fraction S_f of all the pages is guaranteed to be in the invalid/erase state, which increases the odds for the GC algorithm to locate a block with a small number of valid pages (which in turn reduces the WA). In fact, if the SSD has been operational for a while and the TRIM command is not supported (as in older SSDs) or used, the fraction of pages in the invalid/erase state is exactly equal to S_f. This is due to the fact that while the pages corresponding to a file that is deleted by the file system may change their state from valid to invalid, such a change does not occur because file delete information is not by default passed down to the SSD, unless explicitly done so by invoking the TRIM command. In this paper, as in all prior studies, we assume that the TRIM command is not supported or used. Also, the trace data used in Section 5 does not contain any information on the TRIM command either and consists of reads and writes only.

2.2. The d-choices GC algorithm with asymmetry and memory

In this paper we introduce and analyze the d-left and d-memory GC algorithm under both uniform random writes and trace-based workloads. The first algorithm exploits asymmetry, while the latter uses memory.

2.2.1. d-left GC algorithm

The d-left GC algorithm partitions the set of N blocks into K partitions, labeled 1 to K, such that the k-th partition contains N_k blocks, for $K \geq 1$, $k = 1, \dots, K$ and $\sum_k N_k = N$. Further, the d-left GC algorithm chooses d_k blocks uniformly at random from the k-th partition, for $k = 1, \dots, K$, with $d = \sum_k d_k$, and selects the block with the fewest number of valid pages among the d chosen blocks. Ties between blocks belonging to different partitions are broken by selecting the block belonging to the partition with the lowest partition number, while ties between blocks belonging to the same partition are broken uniformly at random.

The d-choices GC algorithm introduced in [25, 14] corresponds to the case with $K = 1$, while for the numerical examples we will restrict ourselves to the case where $N_k = N/d$ and $d_k = 1$, for $k = 1, \dots, K$. The mean field model of the next section however does not impose a restriction on the fractions N_k/N

(or d_k for that matter). Optimizing the fractions N_k/N probably allows us to further reduce the write amplification by a small fraction.

2.2.2. d-memory GC algorithm

The d-memory GC algorithm operates as follows: it stores the id of $c \geq 1$ blocks at all times, these c block ids are initially selected at random and their corresponding blocks are called the c *stored* blocks. Whenever the d-memory GC algorithm is activated, it chooses d blocks uniformly at random from the $N-c$ remaining blocks. Next, the GC algorithm selects the block with the least number of valid pages among the d chosen and the c stored blocks. Finally, the c ids of the blocks with least number of valid pages among the remaining $d+c-1$ blocks become the c stored block ids. In other words, the d-memory GC algorithm stores the id of the 2nd to $(c+1)$st best blocks for possible future use as they might also contains a limited number of valid pages.

3. Mean field models

In this section we generalize the mean field model introduced in [25] in two different ways to study the d-left and d-memory GC algorithm. Under a uniform random writes workload, meaning all pages are accessed equally often and there is no temporal or spacial locality, it is not hard to see that the single and double write frontier achieve the same write amplification[1]. As such we can focus on the single write frontier case only.

3.1. d-left model

When $K = 1$ the model introduced in this section reduces to the one in [25] for the d-choices GC algorithm. Let $\tilde{X}^N_{n,k}(t) \in \{0, 1, \ldots, b\}$, for $n = 1, \ldots, N_k$ and $k = 1, \ldots, K$, be the number of valid pages on block number n of partition k at time t, i.e., just prior to the time epoch where the d-left GC algorithm is activated for the t-th time. Let $\tilde{S} = \{(i,k) | i \in \{0, \ldots, b\}, k \in \{1, \ldots, K\}\}$ and $\tilde{M}^N(t)$ be the occupancy measure of $\tilde{X}^N_{n,k}(t)$, that is, $\tilde{M}^N(t) = \{\tilde{M}^N_{i,k}(t) | (i,k) \in \tilde{S}\}$ and

$$\tilde{M}^N_{i,k}(t) = \frac{1}{N} \sum_{n=1}^{N_k} 1[\tilde{X}^N_{n,k}(t) = i],$$

for $(i,k) \in \tilde{S}$ and where $1[A] = 1$ if A is true and 0 otherwise.

Let $\tilde{\Delta}^N = \{\vec{m} \in \mathbb{R}^{(b+1)K} | m_{i,k}N \in \{0, \ldots, N_k\}, (i,k) \in \tilde{S}, \sum_{i \in S} m_{i,k} = N_k/N\}$ and let $\tilde{p}_{i,k}(\vec{m})$, for $(i,k) \in \tilde{S}$, be the probability that the d-left GC algorithm selects a block with i valid pages that belongs to partition k provided

[1]In the presence of hot and cold data the single and double write frontier create blocks with a different mixture of hot and cold data, which results in a different write amplification as blocks with more hot data are invalidated more easily. Under uniform random writes, both simply create full blocks with data that is equally often accessed. As such the GC algorithm samples from the same distribution when selecting d blocks

that $\tilde{M}^N(t) = \vec{m}$ with $\vec{m} \in \tilde{\Delta}^N$. For the d-left GC algorithms introduced in Section 2.2 we have the following expression for $\tilde{p}_{i,k}(\vec{m})$:

$$\tilde{p}_{i,k}(\vec{m}) = \left(\prod_{s=1}^{k-1} \left(\sum_{j=i+1}^{b} m_{j,s} \right)^{d_s} \right) \cdot$$

$$\left[\left(\sum_{j=i}^{b} m_{j,k} \right)^{d_k} - \left(\sum_{j=i+1}^{b} m_{j,k} \right)^{d_k} \right] \cdot \left(\prod_{s=k+1}^{K} \left(\sum_{j=i}^{b} m_{j,s} \right)^{d_s} \right),$$

as in the first $k-1$ partitions all the chosen blocks should hold at least $i+1$ valid pages, the block with the least number of valid pages among the d_k blocks chosen in partition k has to contain exactly i valid pages and all the chosen blocks in the remaining partitions must contain at least i valid pages.

Define the drift $\vec{f}^N_{left}(\vec{m})$ for $\vec{m} \in \Delta^N$ as the expected change to $\tilde{M}^N(t)$ in one transition, that is,

$$\vec{f}^N_{left}(\vec{m}) = \mathbb{E}[\tilde{M}^N(t+1) - \tilde{M}^N(t) | \tilde{M}^N(t) = \vec{m}].$$

Let $\vec{f}^N_{left}(\vec{m}) = \{\tilde{f}^N_{i,k}(\vec{m}) | (i,k) \in \tilde{S}\}$ and define $\tilde{f}_{i,k}(\vec{m}) = \lim_{N\to\infty} \tilde{f}^N_{i,k}(\vec{m})/\epsilon(N)$, with $\epsilon(N) = 1/N$, then analogously to [25, Section 5.1] one finds for $i < b$

$$\tilde{f}_{i,k}(\vec{m}) = \left(\sum_{j=1}^{b} \sum_{k=1}^{K} \tilde{p}_{b-j,k}(\vec{m}) j \right) \frac{(i+1)m_{i+1,k} - i m_{i,k}}{b\rho} - \tilde{p}_{i,k}(\vec{m}), \qquad (1)$$

while

$$\tilde{f}_{b,k}(\vec{m}) = \sum_{j=0}^{b-1} \tilde{p}_{j,k}(\vec{m}) - \left(\sum_{j=1}^{b} \sum_{k=1}^{K} \tilde{p}_{b-j,k}(\vec{m}) j \right) \frac{b m_{b,k}}{b\rho}. \qquad (2)$$

Intuitively, the expression for $\tilde{f}_{i,k}(\vec{m})$, with $i < b$, can be understood as follows. First, the sum $\sum_{j=1}^{b} \sum_{k=1}^{K} \tilde{p}_{b-j,k}(\vec{m}) j$ represents the mean number of external write requests that are performed in between two executions of the GC algorithm (given that $M(t) = \vec{m}$). Further, $i m_{i,k}/(b\rho)$ is the probability that such an external request updates a valid page on a block of partition k containing i valid pages, due to the uniform random writes and the fact that on average each block holds $b\rho$ valid pages. Hence, with probability $i m_{i,k}/(b\rho)$ the number of blocks containing i valid pages reduces by one, while with probability $(i+1)m_{i+1,k}/(b\rho)$ such a block is created by an external write request. Finally, the number of blocks containing i valid pages also reduces by one in between two executions of the GC algorithm, if the GC algorithm selects such a block (which happens with probability $\tilde{p}_{i,k}(\vec{m})$ if $M(t) = \vec{m}$). The expression for $\tilde{f}_{b,k}(\vec{m}, j)$ can be understood similarly by noting that an extra block in partition k with b valid pages is created (in the WF) whenever the d-left GC algorithm selects a block of partition k containing less than b valid pages.

Define $\hat{M}^N(\tau)$ as the re-scaled process such that $\hat{M}^N(t) = \tilde{M}^N(\lfloor tN \rfloor)$, for $t \geq 0$ and the deterministic process $\vec{\nu}(t) = \{\nu_{i,k}(t)|(i,k) \in \tilde{S}\}$ as the unique solution of the ODE given by

$$\frac{d\vec{\nu}(t)}{dt} = \vec{f}_{left}(\vec{\nu}(t)), \tag{3}$$

where $\vec{f}_{left}(\vec{m}) = \{\tilde{f}_{i,k}(\vec{m})|(i,k) \in \tilde{S}\}$ is defined by (1) and (2). It is not hard to verify that the following theorem holds due to Corollary 1 in [2]:

Theorem 1. *If $\tilde{M}^N(0) \to \vec{m}$ in probability as N tends to infinity, then $\sup_{0 \leq t \leq T} ||\hat{M}^N(t) - \vec{\nu}(t)|| \to 0$ in probability, where $\vec{\nu}(t)$ is the unique solution of the ODE (3) with $\vec{\nu}(0) = \vec{m}$.*

Hence, for N large we can approximate $\tilde{M}^N(t)$ by $\vec{\nu}(t/N)$ for any finite t. Further, Corollary 2 in [2] shows that the convergence extends to the stationary regime if the ODE has a global attractor in $\tilde{\Delta} = \{\vec{m} \in \mathbb{R}^{(b+1)K}|0 \leq m_{i,k} \leq N_k/N, (i,k) \in \tilde{S}, \sum_{i \in S} m_{i,k} = N_k/N\}$. Numerical experiments indicate that there exists a global attractor, but we have no proof at this stage (except for the d-choices GC algorithm with $d = 1$ or $b = 2$).

To determine the write amplification WA of the d-left GC algorithm, we use Euler's method to find a fixed point $\vec{\zeta}$ of (3) with $\nu_{i,k}(0) = \frac{1}{K}\binom{b}{i}\rho^i(1-\rho)^{b-i}$ and a step size $h = 0.001$ until $||\vec{\nu}(t+h) - \vec{\nu}(t)||_1 < 10^{-10}$. For all the experiments conducted convergences occurred within seconds (even for $b = 64$ and $K = 20$). The ODE-based write amplification (WA) is subsequently computed as

$$WA = \frac{b}{b - \sum_{j=1}^{b} j \left(\sum_{K=1}^{K} \tilde{p}_{j,k}(\vec{\zeta})\right)}.$$

3.2. d-memory model

When $c = 0$ the model introduced in this section reduces to the one in [25] for the d-choices GC algorithm. We observe the system just prior to the time epochs where the d-memory GC algorithm is executed and let $X_n^N(t) \in \{0, 1, \ldots, b\}$, for $n = 1, \ldots, N$, be the number of valid pages on block number n at time t for an SSD with N physical blocks. Let $M^N(t) = (M_0^N(t), \ldots, M_b^N(t))$ be the occupancy measure of $X_n^N(t)$, that is,

$$M_i^N(t) = \sum_{n=1}^{N} 1[X_n^N(t) = i],$$

for $i = 0, \ldots, b$. Further, let $j_i(t)$ denote the number of valid pages in the i-th stored block at time t, for $i = 1, \ldots, c$, and let $J^N(t)$ represent the string $j_1(t)j_2(t)\ldots j_c(t)$ and denote $f(J^N(t)) = \min_{i=1}^{c} j_i(t)$. Clearly, in case of uniform random writes $\{(M^N(t), J^N(t)), t \geq 1\}$ is a Markov chain on the state space $\Delta^N \times \mathcal{S}_c$, where $\Delta^N = \{\vec{m} \in \mathbb{R}^{b+1}|m_iN \in \{0, 1, \ldots, N\}, \sum_{i=0}^{b} m_i = 1, \sum_{i=0}^{b} im_i = \rho b\}$ and $\mathcal{S}_c = \{J|J = j_1j_2\ldots j_c, j_i \in \{0, \ldots, b\}, i = 1, \ldots, c\}$, as the total fraction of valid pages equals $\rho = 1 - S_f$ at all times. As this chain cannot be analyzed directly for realistic values of N (e.g., $N = 50,000$), we focus on the system behavior as N tends to infinity using a mean field model, the accuracy of which is validated via simulation in Section 3.3.

Define $f_i^N(\vec{m}, j)$, for $i = 0, \ldots, b$, as the drift of the fraction of blocks with i valid pages provided that the least number of valid pages in one of the stored blocks is j and the occupancy measure is given by $\vec{m}$:

$$f_i^N(\vec{m}, j) = \mathbb{E}[M^N(t+1) - M^N(t) | M^N(t) = \vec{m}, f(J^N(t)) = j].$$

Further, let $f_i(\vec{m}, j) = \lim_{N\to\infty} N f_i^N(\vec{m}, j)$. In order to determine the drift $f_i(\vec{m}, j)$, define $p_i(\vec{m}, j)$ as the probability that the d-memory GC algorithm selects a block with i valid pages provided that $f(J(t)) = j$ and $M(t) = \vec{m}$. These probabilities can be expressed as

$$p_i(\vec{m}, j) = \left(\sum_{\ell=i}^{b} m_\ell\right)^d - \left(\sum_{\ell=i+1}^{b} m_\ell\right)^d, \tag{4}$$

for $i < j$, as all of the d chosen blocks should contain at least i valid pages, but not all should contain more than i valid pages. Further,

$$p_j(\vec{m}, j) = \left(\sum_{\ell=j}^{b} m_\ell\right)^d, \tag{5}$$

while for $i > j$ we clearly have $p_i(\vec{m}, j) = 0$ as at least one of the stored blocks contains only j valid pages.

The reasoning in [25, Section 5.1] to determine the drift $f_i(\vec{m})$ for the d-choices GC algorithm without memory, can now be repeated to express $f_i(\vec{m}, j)$ if we replace the probabilities $p_i(\vec{m})$ in [25] by the probabilities $p_i(\vec{m}, j)$ and by noting that $p_i(\vec{m}, j) = 0$ for $i > j$. Hence based on [25], for $i < b$, we may conclude

$$f_i(\vec{m}, j) = \left(\sum_{\ell=1}^{b} p_{b-\ell}(\vec{m}, j)\ell\right) \frac{(i+1)m_{i+1} - i m_i}{b\rho} - p_i(\vec{m}, j), \tag{6}$$

while

$$f_b(\vec{m}, j) = 1 - p_b(\vec{m}, j) - \left(\sum_{\ell=1}^{b} p_{b-\ell}(\vec{m}, j)\ell\right) \frac{b m_b}{b\rho}. \tag{7}$$

The expressions for $\tilde{f}_{i,k}(\vec{m})$ can be understood intuitively using a similar argument as for (1) and (2), except that we now condition on having $M(t) = \vec{m}$ and $f(J(t)) = j$, while $\sum_{\ell=1}^{b} p_{b-\ell}(\vec{m}, j)\ell$ represents the mean number of external write requests that are performed in between two executions of the GC algorithm.

To define the ODE that captures the evolution of the mean field model, note that $\mathcal{S}_c$ is the state space of $J(t)$. If we assume that $M(t) = \vec{m}$ for $t \geq 0$ for some $\vec{m} \in \Delta$ fixed, then $J(t)$ forms a Markov chain with state space $\mathcal{S}_c$. Denote its transition matrix of size $(b+1)^c$ as $K(\vec{m})$, as it depends on the occupancy measure $\vec{m}$. It is not hard to see that $K(\vec{m})$ contains a single recurrent class (for $d > 1$) that consists of all the strings $J = j_1 j_2 \ldots j_c$ for which $m_{j_i} > 0$ for $i = 1, \ldots, c$. Denote $\pi(\vec{m}) = (\pi_J(\vec{m}))_{J \in \mathcal{S}_c}$ as the invariant vector of $K(\vec{m})$ (with the entries of the possible transient states set to zero). Next, we define the set of ODEs that captures the evolution of the mean field model as

$$\frac{d\vec{\mu}(t)}{dt} = \vec{F}(\vec{\mu}(t)), \tag{8}$$

with

$$\vec{F}(\vec{m}) = \sum_{j=0}^{b} \left(\sum_{J \in \mathcal{S}_c, f(J)=j} \pi_J(\vec{m}) \right) \vec{f}(\vec{m}, j).$$

Hence, the ODE is expressed via the drifts $\vec{f}(\vec{m}, j) = (f_0(\vec{m}, j), \ldots, f_b(\vec{m}, j))$ and the probabilities $\pi_j(\vec{m})$ defined as $\pi_j(\vec{m}) = \sum_{J \in \mathcal{S}_c, f(J)=j} \pi_J(\vec{m})$, for $j = 0, \ldots, b$.

Finally, define $\bar{M}^N(\tau)$ as the re-scaled process such that $\bar{M}^N(t) = M^N(\lfloor tN \rfloor)$. As in [25], it is not hard to verify that the following theorem holds due to Corollary 1 in [2]:

Theorem 2. *If $M^N(0) \to \vec{m}$ in probability as N tends to infinity, then $\sup_{0 \leq t \leq T} ||\bar{M}^N(t) - \vec{\mu}(t)|| \to 0$ in probability, where $\vec{\mu}(t)$ is the unique solution of the ODE (8) with $\vec{\mu}(0) = \vec{m}$.*

Hence, as for the d-left GC algorithm, we can approximate $M^N(t)$ by $\vec{\mu}(t/N)$ for any finite t for N large. To show that the convergence extends to the stationary regime, it suffices to prove that the ODE has a global attractor. Unfortunately proving the existence of a global attractor for the set of ODEs in (8) appears hard, but numerical experiments suggest that such an attractor exists.

To determine the write amplification WA, we once more use Euler's method to find a fixed point $\vec{\eta} = (\eta_0, \ldots, \eta_b)$ of (8) and compute the write amplification as

$$WA = \frac{b}{b - \sum_{j=0}^{b} \pi_j(\vec{\eta}) \sum_{i=0}^{j} i p_i(\vec{\eta}, j)}.$$

Note Euler's method is an iterative method and when applied to (8) this implies that we need to determine the steady state probabilities of the transition matrix $K(\vec{m})$ for some $\vec{m}$ during each step (while the number of steps can be as large as a few thousand). As $K(\vec{m})$ is of size $(b+1)^c$, we cannot simply construct $K(\vec{m})$ and solve the corresponding linear system to determine the necessary steady state probabilities $\pi_j(\vec{m})$ for realistic values of b and c, e.g., $b = 64$ and $c = 10$. In the next subsection we show how we can reduce the problem of computing the probabilities $\pi_j(\vec{m})$, which is required during each step of Euler's method, to the computation of the steady state probabilities of b small Markov chains (each having $c + 1$ states).

3.2.1. Computation of $\pi_j(\vec{m})$

To determine the probabilities $\pi_j(\vec{m})$, let $j \in \{0,\dots,b-1\}$ be *fixed* and denote $A_{j,k} \subset \mathcal{S}_c$, for $k = 0,\dots,c$, as the subset of $\mathcal{S}_c$ containing all the strings $J \in \mathcal{S}_c$ such that exactly k elements of J are larger than j. Clearly $A_{j,0},\dots,A_{j,c}$ is a partition of $\mathcal{S}_c$ and we show that the Markov chain with transition matrix $K(\vec{m})$ can be lumped with respect to this partition. In other words, the number of stored blocks with more than j valid pages forms a $(c+1)$-state Markov chain with state space $\{0,\dots,c\}$ for any $j \in \{0,\dots,b-1\}$ fixed. Denote its transition matrix as $P^{(j)}$ and its transition probabilities as $p^{(j)}_{i,i'}$ with $i, i' \in \{0,\dots,c\}$, then

$$p^{(j)}_{i,i-k}(\vec{m}) = \begin{cases} B^{d,\sum_{\ell=0}^{j} m_\ell}_{k+1} & i < c; -1 \le k < i, \\ B^{d,\sum_{\ell=0}^{j} m_\ell}_{k+1} & i = c; 1 \le k < c, \\ \sum_{s=0}^{1} B^{d,\sum_{\ell=0}^{j} m_\ell}_{s} & i = c; k = 0, \\ 1 - \sum_{s=0}^{i} B^{d,\sum_{\ell=0}^{j} m_\ell}_{s} & k = i, \\ 0 & \textit{otherwise} \end{cases}$$

where $B^{n,p}_j = \binom{n}{j} p^j (1-p)^{n-j}$. Indeed, if at least one stored block contains at most j valid pages (i.e., if $i < c$), the number of stored blocks with more than j valid pages increases by one during a single transition if none of the d chosen blocks contains at most j valid pages. Similarly, if $i < c$, the state remains the same if exactly one of the d chosen blocks contains at most j valid pages and it decreases by k (for $k < i$) if $k+1$ of the chosen blocks contain at most j valid pages. The expressions for the cases with $i = c$ and $k < c$ can be understood similarly. Finally, a transition to state 0 occurs as soon as more than i of the chosen d blocks contain at most j valid pages.

Denote the steady state probability vector of the $(c+1)$-state Markov chain with transition matrix $P^{(j)}$ as $(\theta^{(j)}_0(\vec{m}),\dots,\theta^{(j)}_c(\vec{m}))$. These vectors can be computed easily (in fact they can even be expressed explicitly in a recursive manner as this chain is skip-free in one direction) and they obey the following equalities:

$$\sum_{\ell=0}^{j} \pi_\ell(\vec{m}) = \sum_{J \in \mathcal{S}_c, f(J) \le j} \pi_J(\vec{m}) = 1 - \theta^{(j)}_c(\vec{m}),$$

as in any state $i < c$ there is at least one stored block with at most j valid pages. If we compute the steady state probabilities $\theta^{(j)}_c(\vec{m})$, for $j = 0,\dots,b-1$, by solving b Markov chains with $c+1$ states each, the above equality allows us to express the probabilities $\pi_j(\vec{m})$ as

$$\pi_j(\vec{m}) = \theta^{(j-1)}_c(\vec{m}) - \theta^{(j)}_c(\vec{m}),$$

for $j = 1,\dots,b-1$, while $\pi_0(\vec{m}) = 1 - \theta^{(0)}_c(\vec{m})$ and $\pi_b(\vec{m}) = \theta^{(b-1)}_c(\vec{m})$. To conclude, we can compute the required probabilities $\pi_j(\vec{m})$, for $j = 0,\dots,b$, by solving b Markov chains of size $c+1$ only, instead of trying to solve the $(b+1)^c$-state Markov chain characterized by $K(\vec{m})$. This allows us to generate

b	S_f	d	ODE	simul. (95% conf.)
64	0.07	5	7.4042	7.4040 $\pm$ 0.0010
64	0.14	12	3.6569	3.6570 $\pm$ 0.0002
64	0.21	8	2.5933	2.5932 $\pm$ 0.0001
32	0.08	10	5.7228	5.7229 $\pm$ 0.0004
32	0.13	3	4.5260	4.5262 $\pm$ 0.0007
32	0.18	20	2.7861	2.7860 $\pm$ 0.0001
16	0.06	14	6.1242	6.1246 $\pm$ 0.0005
16	0.13	7	3.6185	3.6187 $\pm$ 0.0004
16	0.20	4	2.7597	2.7596 $\pm$ 0.0004

Table 1: Comparison of ODE-based write amplification and simulation experiments (25 runs) for the d-left GC algorithm with $N/d = 5,000$ blocks. Relative errors are less than 0.05%

b	S_f	d	c	ODE	simul. (95% conf.)	nruns
64	0.08	5	2	6.2461	6.2468 $\pm$0.0006	100
64	0.12	6	24	4.2408	4.2405 $\pm$0.0005	50
64	0.17	8	8	3.0596	3.0595 $\pm$0.0003	25
32	0.07	6	5	6.4146	6.4147 $\pm$0.0007	100
32	0.11	20	3	4.2113	4.2114 $\pm$0.0006	50
32	0.16	15	19	3.0668	3.0664 $\pm$0.0004	25
16	0.06	10	1	6.1340	6.1346 $\pm$0.0010	100
16	0.10	4	10	4.5355	4.5344 $\pm$0.0011	50
16	0.15	2	3	3.9448	3.9447 $\pm$0.0017	25

Table 2: Comparison of ODE-based write amplification and simulation experiments for the d-memory GC algorithm with $N = 50,000$ blocks. Relative errors are less than 0.05%.

numerical results in a matter of seconds with very low memory requirements even for systems with $b = 64$ pages per block that rely on $c = 50$ memory locations.

3.3. Validation

To validate both mean field models for uniform random writes we conducted simulation experiments that basically simulate the Markov chains $\{\tilde{M}^N(t), t \geq 1\}$ and $\{(M^N(t), J^N(t)), t \geq 1\}$. Hence, the only difference with the mean field model is therefore the system size N, which was set equal to 50,000 for the d-memory GC algorithm and 5,000 times d for the d-left GC algorithm in the simulation setup. For $b = 64$ and 4 KB pages this corresponds to an SSD with a total storage capacity of 12.8 GB and 1.28 times d GB, respectively.

The number of runs used in the simulation setup to compute the 95% confidence intervals was 25 for the d-left GC algorithm and varied between 25 and 100 depending on the spare factor S_f for the d-memory GC algorithm (see Table 2). Each run had a length of 250,000 for the d-memory GC algorithm and of 150,000 times d for the d-left GC algorithm . The warm-up period was set equal to 1/3th of the total length. The results in Table 1 and Table 2 show

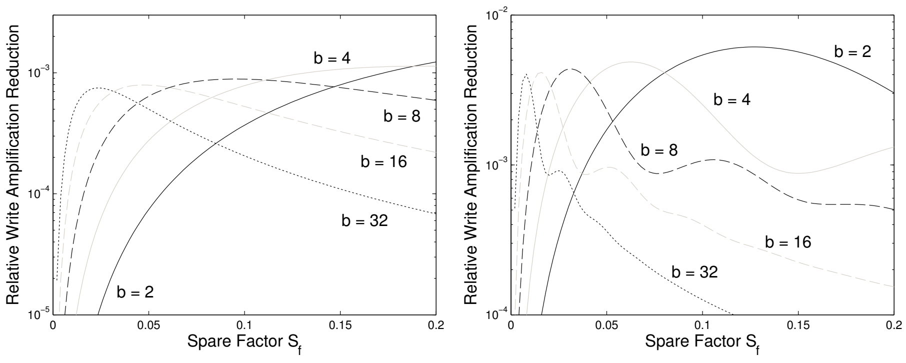


Figure 1: Relative reduction in the write amplification by the d-left GC algorithm for $d = 2$ (left) and $d = 10$ (right). All gains are below 1%.

a good agreement between the model and the simulation results with relative errors below 0.05% for some arbitrary combinations of b, c, d and S_f.

4. Uniform random writes: numerical results

4.1. d-left GC algorithm

In this section we rely on the mean field model of Section 3.1 to study the impact of having asymmetry in case of uniform random writes. More specifically, in Figure 1 we compare the write amplification of the d-left and d-choices GC algorithm for $d = 2$ and $d = 10$. In fact, in all the numerical experiments the d-left algorithm reduced the write amplification of the d-choices algorithm and as such we have plotted the relative reduction in the write amplification. The results show that the gain is very limited, i.e., below 1%, and is quite sensitive to the number of pages b per block, the spare factor $S_f = 1 - \rho$ and the number of choices d. Similar results were also obtained for other d values.

Although there is little practical interest in a system with very small b values, e.g., $b = 4$, we included such results as they reveal a pattern in the impact of b. The fact that these curves become quite irregular as d becomes larger seems to be related to the fact that the write amplification of the d-choices GC algorithm approaches the write amplification of the greedy GC algorithm under uniform random writes (see [25]) and the write amplification of the latter GC algorithm is not smooth in S_f [3].

Figure 2 shows the distribution of the number of valid pages per block for $b = 16$, $S_f = 0.1$ and $d = 4$ for the d-choices and d-left GC algorithm. It indicates that the partitions with a higher label number are more likely to contain less valid pages. This can be easily understood by noting that blocks repeatedly become full (when selected by the GC algorithm) and subsequently lose valid pages until they are selected again. As the blocks belonging to the lower partition numbers are selected in case of a tie, the blocks in the higher partitions have more time to loose valid pages. In comparison with the d-choices

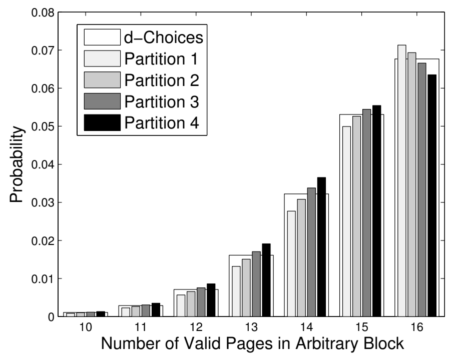


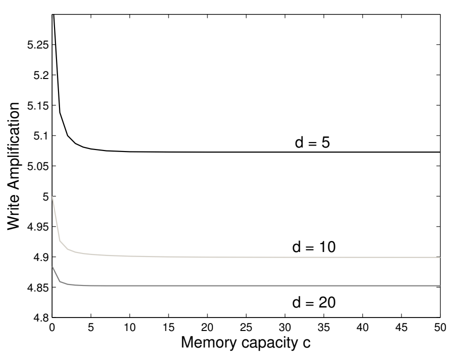
Figure 2: Distribution of the number of valid pages for $d = 4$, $b = 16$ and $S_f = 0.1$.

algorithm the blocks belonging to partition 3 and 4 are about 7 and 23% more likely to contain a small number of valid pages (in this example), respectively, while partitions 1 and 2 are somewhat less likely to contain a small number of valid pages. This asymmetry turns out to give the d-left GC algorithm a very minor edge on the d-choices GC algorithm.

4.2. d-memory GC algorithm

In this section we rely on the mean field model of Section 3.2 to look at the impact of introducing memory on the write amplification under uniform random writes. In the first experiment we increase c while all the other parameters, including d, remain fixed. The write amplification should reduce as c increases and one may expect that the write amplification approaches the write amplification of the greedy algorithm as c becomes large, because setting $c = N$ corresponds to the greedy GC algorithm. This might in fact be the case if we were to define c as a function of N and let N tend to infinity. We should however keep in mind that in our setup c remains fixed as N tends to infinity and therefore the write amplification does not necessarily reduce to the one of the greedy GC algorithm as c tends to infinity.

The results in Figure 3(left), where $b = 64$, $S_f = 0.1$ and $d = 5, 10$ and 20, confirm that the write amplification decreases as c increases, but the rate of decrease very quickly diminishes as c increases and seems to stagnate rather quickly. The write amplification of the greedy algorithm equals 4.8213 for $b = 64$ and $S_f = 0.1$, which seems to suggest that the write amplification does not converge to the one of the greedy algorithm as c tends to infinity (with d fixed). Figure 3(right) seems to confirm this as it suggests that the distribution of the least number of valid pages in a stored block converges to a distribution that is clearly different from the distribution of the number of valid pages in a block selected by the greedy algorithm (computed via [3]). In conclusion, adding a very limited amount of memory to reduce the write amplification is quite effective, e.g., $c \leq 5$, but the additional benefit of adding larger amounts of



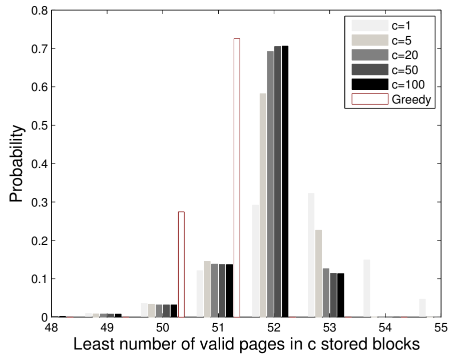
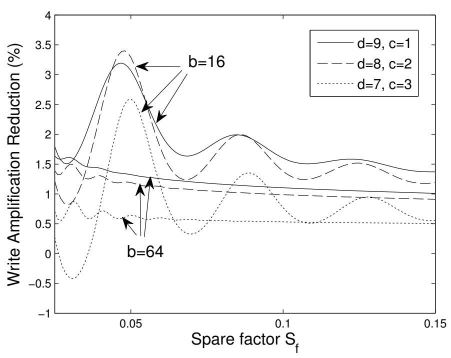


Figure 3: Impact of memory on the write amplification of the d-memory GC algorithm for $b = 64$ and $S_f = 0.1$ (left). Comparison of the distribution of $\pi_j(\vec{\eta})$, the least number of valid pages in memory, for $b = 64$, $d = 10$ and $S_f = 0.1$ and various c values with the distribution of the greedy GC algorithm (right).



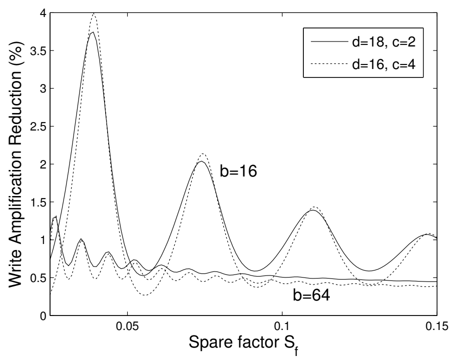


Figure 4: Reduction in the write amplification (%) of the d-choices algorithm with memory when $c > 0$ compared to the case with $c = 0$, for $c + d = 10$ (left) and $c + d = 20$ (right).

memory is far less significant. Similar results were obtained for other parameter settings.

In the second experiment we intend to see whether it is best to reselect d new blocks uniformly at random each time the GC algorithm is executed or whether to reselect only $d - c$ of the blocks and to store the ids of the c best blocks in memory. In other words, we fix b, S_f and $c + d$, while we increase c. Figure 4 depicts the relative gain of the d-choices algorithm with memory when $c > 0$ compared to the case with $c = 0$. The left plot shows that setting $c = 1$ is optimal for $c + d = 10$, except for $b = 16$ and spare factors around 0.05. The gains compared to setting $d = 10$ are quite limited and typically lie between 1 and 3 percent. More substantial gains are observed when we perform a similar experiment using trace-based data in Section 5.2. For $c + d = 20$ we observe similar results, meaning moderate gains are obtained if we do not always reselect

d new blocks uniformly at random. The oscillations observed in the curves in Figure 4 are related to the fact that the write amplification of the d-choices GC algorithm approaches the amplification of the greedy GC algorithm as d tends to infinity and this latter write amplification is not a smooth function of S_f. The oscillations also become more pronounced as d increases.

5. Trace-based workloads

5.1. Simulation setup

When conducting (time-consuming) trace-based simulation experiments we have relied on a number of real-world I/O traces:

- **rsrch0** [22]: an I/O trace collected at a server supporting research projects at Microsoft Research.
- **prxy0** [22]: an I/O trace containing requests of a Firewall/web proxy server at Microsoft Research.
- **online** [27]: an I/O trace of a coursework management workload on Moodle at a university.
- **webmail** [27]: an I/O trace of webmail traffic on a university department mail server.

In order to use these traces in an SSD setting with 4 KB pages the traces were processed as follows. We first aligned the offset of each request to a multiple of 4 KB (all the offsets in the traces are multiples of 512 bytes). Requests with sizes above 4 KB (if present) were subsequently split into several (sequential) requests such that all requests have a size of at most 4 KB. Some statistics on the trace files after this processing was done are listed in Table 3 and Table 4. Table 3 lists the percentage of the requests that are write requests (%Writes), it lists the number of requests (#Requests), and the percentage of the accessed logical block address (LBA) space that is only accessed by read requests (%LBA_RO). The data locality, presented in Table 4, is expressed as follows: the y% most frequently accessed pages represent v% percent of the total accessed LBA space (including both writes and reads) if v appears in the column labeled y%, e.g., for the webmail trace the 60% most frequently accessed pages represent about 5.23% percent of the total accessed LBA space. Table 3 also shows that all these traces are write dominant, although in some cases (i.e., the webmail and online trace) a fairly large portion of the accessed LBA space is accessed only by read requests.

The SSD used in the trace-driven simulation experiments is composed of $bU = b\lfloor x/b \rfloor$ logical pages, where x is equal to the number of logical pages accessed during the trace. In other words, we have remapped the accessed logical block addresses into this smaller range (instead of using the span of the trace to determine the SSD volume size). The number of physical blocks N is determined by U by means of the spare factor S_f, i.e., $U = N(1 - S_f)$. Thus,

I/O trace	%Writes	#Requests	%LBA_RO
rsrch0 [22]	88.87	3,253,639	19.02
prxy0 [22]	96.36	22,136,692	19.53
online [27]	73.88	5,700,499	64.87
webmail [27]	81.86	7,795,815	55.19

Table 3: Data set statistics. %Writes: percentage of writes, #Requests: number of request and %LBA_RO: the size of the LBA space that is only read.

I/O trace	Data Locality			
	20%	40%	60%	80%
rsrch0 [22]	0.01	0.09	7.15	19.17
prxy0 [22]	0.06	0.12	0.79	5.20
online [27]	1.30	7.76	14.30	30.79
webmail [27]	0.10	0.44	5.23	16.16

Table 4: Data set statistics: LBA locality.

a fraction $(1 - S_f)$ of the pages is in the valid state at all times during the simulation, while all of the logical pages are accessed at least once during the simulation (except for at most $b - 1$ pages), but data is only written to some of the pages. The data is initially placed in an unfragmented manner on the drive such that the first U physical blocks contain all the valid pages and the remaining $N - U$ blocks contain only pages in the erase state.

The initial state of the drive has, in some cases, an important impact on the results in this type of simulation setup. To understand this it is important to note that in some traces up to 50% of the logical address space is accessed by read operations only. If b pages that are only read during the simulation happen to reside on the same physical block at the start of the simulation, they will still reside there at the end of the simulation. Thus, if a substantial fraction of the blocks is full of valid data that is only read, say a fraction F, the spare space (of size S_f) is fragmented over a smaller part of the physical address space and the *effective* spare factor is more like $S_f/(1 - F)$; hence we expect to see lower values for the write amplification as F increases. As the initialization method used in our experiments starts with unfragmented data located at the first U blocks of the disk, we will observe lower values for the write amplification for larger %LBA_RO values, i.e., for the online and webmail trace.

Finally, to make the simulation runs sufficiently large we also adopted the *replay* method used in prior SSD work [14, 21]. By replaying a trace, we simply mean that the I/O pattern of the trace is repeated a number of times without change such that the overall trace length exceeds $50,000,000$ requests. This implies that pages are updated several times during a single run, unless the page is only read during the original trace. The results in Figure 5 are based on 5 to 12 runs (depending on the trace) such that the 95% confidence intervals are sufficiently small. The results in Tables 5, 6 and 7 are based on 10 runs.

S_f	Single WF		Double WF	
	d-choices	d-left (gain)	d-choices	d-left (gain)
rsrch0 trace				
0.14	2.843	2.843 (0.00%)	1.785	1.781 (0.23%)
0.10	3.739	3.738 (0.05%)	2.095	2.080 (0.72%)
0.06	5.601	5.602 (-0.02%)	2.826	2.785 (1.47%)
prxy0 trace				
0.14	3.330	3.329 (0.03%)	1.363	1.355 (0.59%)
0.10	4.258	4.257 (0.03%)	1.623	1.608 (0.94%)
0.06	6.105	6.108 (-0.05%)	2.273	2.246 (1.21%)
online trace				
0.14	1.513	1.513 (-0.01%)	1.429	1.410 (1.36%)
0.10	1.907	1.907 (-0.02%)	1.761	1.728 (1.86%)
0.06	2.830	2.833 (-0.11%)	2.506	2.465 (1.63%)
webmail trace				
0.14	1.859	1.860 (-0.07%)	1.430	1.410 (1.43%)
0.10	2.414	2.413 (0.03%)	1.729	1.683 (2.67%)
0.06	3.600	3.601 (-0.02%)	2.441	2.336 (4.30%)

Table 5: Write amplification of d-left and d-choices GC algorithm for trace-based workloads using both a single or double write frontier, with $b = 64$ pages per block and $K = d = 10$.

5.2. Numerical results

5.2.1. d-left GC algorithm

For the trace-based experiments we partitioned the drive in $K = 10$ parts such that block number n belongs to partition k if $n = k \mod K$, for $n = 1, \ldots, N$ and $k = 1, \ldots, K$. In this manner all the partitions contain a similar fraction of the spare space at the start of the simulation run. The number of choices in each partition was set equal to 1, i.e., $d_k = 1$ for $k = 1, \ldots, K$.

The simulation results for each of the four traces with a single and double write frontier are presented in Table 5 for $S_f = 0.06, 0.10$ and 0.14 and $b = 64$. The results for the single WF indicate that incorporating asymmetry seems to have little to no impact on the write amplification. This is very much in line with the findings of the mean field model for uniform random writes. In fact, the simulation results of the d-left and d-choices GC algorithms are so close that it is impossible to state that asymmetry always results in a very minor gain (as indicated by the mean field model in case of uniform random writes).

For the double WF the results are clearer: incorporating asymmetry always results in a gain and the gain is more pronounced compared to the uniform random write setting (see Section 4.1). The gain is however still below 2% except for the webmail trace with a relatively small spare factor.

5.2.2. d-memory GC algorithm

We start by repeating one of the experiments of Section 4.2 where we fixed $b = 64$ and $c + d = 10$ and compared the write amplification for different values of c. The results for $S_f = 0.06, 0.10$ and 0.14 in case of a single write frontier are presented in Table 6, while Table 7 contains the results for the double write

S_f	$c = 0$	$c = 1$ (gain)	$c = 2$ (gain)	$c = 3$ (gain)
rsrch0 trace				
0.14	2.843	2.790 (1.86%)	2.803 (1.38%)	2.834 (0.32%)
0.10	3.739	3.649 (2.40%)	3.663 (2.03%)	3.709 (0.82%)
0.06	5.601	5.446 (2.77%)	5.470 (2.35%)	5.550 (0.92%)
prxy0 trace				
0.14	3.330	3.316 (0.42%)	3.325 (0.14%)	3.343 (-0.40%)
0.10	4.258	4.227 (0.48%)	4.249 (0.22%)	4.275 (-0.41%)
0.06	6.105	6.062 (0.70%)	6.086 (0.32%)	6.138 (-0.54%)
online trace				
0.14	1.513	1.387 (8.30%)	1.394 (7.90%)	1.441 (4.78%)
0.10	1.907	1.753 (8.09%)	1.772 (7.06%)	1.850 (3.00%)
0.06	2.830	2.600 (8.12%)	2.655 (6.19%)	2.809 (0.76%)
webmail trace				
0.14	1.859	1.781 (4.20%)	1.795 (3.46%)	1.836 (1.25%)
0.10	2.414	2.307 (4.43%)	2.327 (3.59%)	2.387 (1.12%)
0.06	3.600	3.412 (5.24%)	3.452 (4.11%)	3.554 (1.27%)

Table 6: Impact of memory on the write amplification using trace based simulations with $b = 64$ and $c + d = 10$: Single write frontier.

frontier as introduced in [26]. A first observation is that the write amplification is reduced quite drastically if we replace the single by the double write frontier. This shows that the double write frontier not only significantly reduces the write amplification in case of a simple Rosenblum workload model as demonstrated in [26] (which contains no spacial locality and the temporal locality does not vary over time), but is equally effective in case of real life workloads.

A second observation is that the largest reduction in the write amplification, in case of the single write frontier, is obtained when $c = 1$ (when compared to setting $c = 0$). A similar observation was made in Figure 4 for $c + d = 10$ under uniform random workloads, where the gain was between 1% and 1.5% for $b = 64$. For real life workloads the gain is much more pronounced, with gains up to 8%. For the double write frontier the gains are even more substantial (up to 13%) and setting $c = 2$ is best in some cases, though the additional gain compared to having $c = 1$ is rather limited in such cases. Given the above results, we focus on SSDs that rely on the double write frontier in the remainder of this section.

The greedy GC algorithm is no longer optimal in case of non-uniform workloads [26]. In fact, the results in [26] showed that under the Rosenblum model, there exists an optimal parameter d for the d-choices GC algorithm (i.e., a choice for d that minimizes the write amplification). We have also analyzed the existence of an optimal value for d for the d-memory GC algorithm for each of the four traces discussed in Section 5.1 and for various choices of c. We only present the results for the rsrch0 and webmail trace, as the results for the prxy0 and online trace are quite similar.

Figure 5 plots the write amplification of the rsrch0 and webmail trace as a function of the number of choices d for various c values with $S_f = 0.1$ and $b = 64$. Looking at the rsrch0 results, we see that for c fixed the curves appear

S_f	$c = 0$	$c = 1$ (gain)	$c = 2$ (gain)	$c = 3$ (gain)
rsrch0 trace				
0.14	1.785	1.690 (5.32%)	1.696 (5.00%)	1.723 (3.48%)
0.10	2.095	1.929 (7.90%)	1.942 (7.31%)	1.993 (4.85%)
0.06	2.826	2.538 (10.2%)	2.578 (8.78%)	2.725 (3.60%)
prxy0 trace				
0.14	1.363	1.232 (9.59%)	1.228 (9.88%)	1.258 (7.68%)
0.10	1.623	1.439 (11.3%)	1.454 (10.5%)	1.551 (4.45%)
0.06	2.273	2.064 (9.20%)	2.187 (3.80%)	2.443(-7.47%)
online trace				
0.14	1.429	1.267 (11.4%)	1.259 (11.9%)	1.301 (8.97%)
0.10	1.761	1.554 (11.7%)	1.556 (11.6%)	1.640 (6.88%)
0.06	2.506	2.237 (10.7%)	2.305 (8.01%)	2.497 (0.35%)
webmail trace				
0.14	1.430	1.273 (11.0%)	1.259 (12.0%)	1.284 (10.2%)
0.10	1.729	1.507 (12.8%)	1.501 (13.2%)	1.580 (8.60%)
0.06	2.441	2.158 (11.6%)	2.243 (8.09%)	2.479 (-1.57%)

Table 7: Impact of memory on the write amplification using trace based simulations with $b = 64$ and $c + d = 10$: Double write frontier.

convex and there exists an optimal value for d, located at $56, 37, 34$ and 37 for $c = 0, 1, 2$ and 3, respectively. Further, the lowest write amplification is realized by setting $c = 1$ and $d = 37$, meaning the optimal d-choices GC algorithm without memory is outperformed by the one with memory, though the gain is very limited. For the webmail trace we observe similar results for $c = 0, 2, 4$ and 6, but the optimal c and d value is larger (i.e., the optimal d is between 100 and 150) and the differences are even less substantial.

These results may seem to indicate that having memory is of little value after all as the optimal d with no memory performs quite similar to the optimal c and d combination. One should however keep in mind that the optimal d value is clearly case specific and hard to determine in practice (and may change over time). A more practical approach may therefore exist in using a fixed d, e.g., $d = 10$ or 20. In such a case adding a limited amount of memory can make a significant difference as shown before. Another thing to note is that having some memory often allows the use of a smaller value of d without increasing the write amplification.

Figure 5 also illustrates that adding more memory, i.e., increasing c, while keeping d fixed, does not always result in a reduction of the write amplification. This is in contrast to the results presented in Figure 3 which corresponded to the synthetic uniform random writes model (under which the performance of the single and double write frontier is identical). In other words, in the presence of hot and cold data there exists an optimal amount of memory when trying to minimize the write amplification by means of the d-memory GC algorithm. Determining this value in practice is again hard and therefore using a limited fixed amount of memory, e.g., $c = 1$ or 2 for $d = 10$, seems like a good compromise given the results in this section.

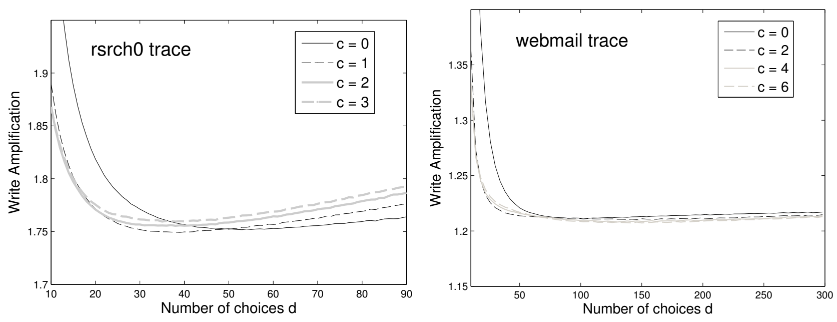


Figure 5: Write amplification as a function of d for various c values and $S_f = 0.1$ and $b = 64$ for the rsrch0 (left) and webmail trace (right).

6. Conclusions

In this paper we introduced the d-choices GC algorithm with asymmetry and memory and analyzed its impact on the write amplification using both synthetic and trace-based workloads. The synthetic workloads consisted of uniform random writes and the resulting write amplification was analyzed via two mean field models, where the main challenge for the system with memory existed in determining the fixed point of the resulting set of ODEs in an efficient manner. The trace-based workloads were analyzed by simulation and considered both SSDs using a single and a double write frontier.

Numerical results for the d-choices GC algorithm with asymmetry indicated that the reduction in the write amplification is typically below 2% in case of the double write frontier, while negligible in case the system operates using a single write frontier.

For the d-choices GC algorithm with memory there exists an optimal combination of c, the number stored memory ids, and d, the number of random choices, in the presence of hot and cold data. However these optimal combinations are hard to determine and very case specific. As such a more practical approach exists in working with a fixed c and d such that one obtains an overall good performance. In such a case selecting a small c value can reduce the write amplification by 10% or more in case of real workloads compared to having no memory at all. Further, adding memory to the d-choices GC algorithm allows one to use smaller d values without increasing the write amplification.

[1] Y. Azar, A.Z. Broder, A.R. Karlin, and E. Upfal. Balanced allocations. In *SIAM Journal on Computing*, pages 593–602, 1994.

[2] M. Benaïm and J. Le Boudec. A class of mean field interaction models for computer and communication systems. *Performance Evaluation*, 65(11-12):823–838, 2008.

[3] W. Bux and I. Iliadis. Performance of greedy garbage collection in flash-based solid-state drives. *Perform. Eval.*, 67(11):1172–1186, November 2010.

[4] F. Chen, D.A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. *ACM SIGMETRICS Perform. Eval. Rev.*, 37(1):181–192, 2009.

[5] T.S. Chung, D.J. Park, S. Park, D.H. Lee, S.W. Lee, and H.J. Song. A survey of flash translation layers. *Journal of Systems Architecture*, 55:332–343, 2009.

[6] P. Desnoyers. Analytic models of SSD write performance. *ACM Trans. Storage*, 10(2):8:1–8:25, March 2014.

[7] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37:138–163, 2005.

[8] L. M. Grupp, J. D. Davis, and S. Swanson. The bleak future of NAND flash memory. In *Proc. of USENIX Conference on File and Storage Technologies*, 2012.

[9] X. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka. Write amplification analysis in flash-based solid state drives. In *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, SYSTOR '09, pages 10:1–10:9, New York, NY, USA, 2009.

[10] J. Kang, H. Jo, J. Kim, and J. Lee. A superblock-based flash translation layer for NAND flash memory. In *In EMSOFT 2006: Proceedings of the 6th ACM IEEE International conference on Embedded software*, pages 161–170. ACM, 2006.

[11] Jesung Kim, Jong Min Kim, Sam H. Noh, Sang Lyul Min, and Yookun Cho. A space-efficient flash translation layer for compactflash systems. *IEEE Transactions on Consumer Electronics*, 48:366–375, 2002.

[12] S. Lee, D. Shin, Y.-J Kim, and J. Kim. LAST: locality-aware sector translation for NAND flash memory-based storage systems. *SIGOPS Oper. Syst. Rev.*, 42(6):36–42, October 2008.

[13] 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 Trans. Embed. Comput. Syst.*, 6(3), July 2007.

[14] 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. *ACM SIGMETRICS Perform. Eval. Rev.*, 41(1):179–190, 2013.

[15] J. Menon. A performance comparison of RAID-5 and log-structured arrays. In *Proceedings of the 4th IEEE International Symposium on High Performance Distributed Computing*, HPDC '95, pages 167–178, Washington, DC, USA, 1995.

[16] J. Menon and L. Stockmeyer. An age-threshold algorithm for garbage collection in log-structured arrays and file systems. In *High Performance Computing Systems and Applications*, volume 478 of *The Springer International Series in Engineering and Computer Science*, pages 119–132. Springer US, 1998.

[17] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Trans. Parallel Distrib. Syst.*, 12:1094–1104, October 2001.

[18] M. Mitzenmacher, B. Prabhakar, and D. Shah. Load balancing with memory. In *In Proc. of the 43rd IEEE Symp. on Foundations of Computer Science (FOCS)*, 2002.

[19] M. Mitzenmacher, A. Richa, and R. Sitaraman. The power of two random choices: a survey of techniques and results. *Handbook of Randomized Computing*, 1, 2001.

[20] M. Mitzenmacher and B. Vöcking. The asymptotics of selecting the shortest of two, improved. In *Analytic Methods in Applied Probability: In Memory of Fridrikh Karpelevich*, pages 165–176, 2002.

[21] M. Murugan and D. Du. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In *In Proc. of IEEE MSST*, 2011.

[22] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. *Trans. Storage*, 4(3):10:1–10:23, November 2008.

[23] J.T. Robinson. Analysis of steady-state segment storage utilizations in a log-structured file system with least-utilized segment cleaning. *SIGOPS Oper. Syst. Rev.*, 30(4):29–32, October 1996.

[24] B. Van Houdt. Analysis of the d-choices garbage collection algorithm with memory in flash-based ssds. In *Proceedings of Valuetools, Torino (Italy)*, DEC 2013.

[25] B. Van Houdt. A mean field model for a class of garbage collection algorithms in flash-based solid state drives. *ACM SIGMETRICS Perform. Eval. Rev.*, 41(1):191–202, 2013.

[26] B. Van Houdt. Performance of garbage collection algorithms for flash-based solid state drives with hot/cold data. *Performance Evaluation*, 70(10):692–703, 2013.

[27] A. Verma, R. Koller, L. Useche, and R. Rangaswami. SRCMap: energy proportional storage using dynamic consolidation. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, pages 267–280, Berkeley, CA, USA, 2010.

[28] B. Vöcking. How asymmetry helps load balancing. *Journal of the ACM*, 50(4):568–589, 2003.

[29] N.D. Vvedenskaya, R.L. Dobrushin, and F.I. Karpelevich. Queueing system with selection of the shortest of two queues: an asymptotic approach. *Problemy Peredachi Informatsii*, 32:15–27, 1996.

[30] L. Xiang and B. Kurkoski. An improved analytical expression for write amplification in NAND flash. In *International Conference on Computing, Networking, and Communications (ICNC)*, pages 497–501, 2012.