How to improve the performance of the d-choices garbage collection algorithm in flash-based SSDs

R. Verschoren and B. Van Houdt
Dept. of Mathematics and Computer Science
University of Antwerp, Belgium

ABSTRACT

The performance of flash-based solid state drives is greatly impacted by the garbage collection algorithm. The d-choices garbage collection algorithm, which selects a victim block with the fewest number of valid pages among d randomly selected blocks, is known to perform well in terms of the write amplification. However, the number of erasures performed on a block may be quite unbalanced, which reduces the lifespan of SSDs that can only tolerate a limited number of erasures per block. This unequal wear is caused by the hot/cold data separation used to achieve a low write amplification, as blocks holding hot data tend to endure more erasures.

Methods to reduce this unequal wear often cause a significant increase in the write amplification (which slows down the device). In this paper we propose a new mechanism that allows us to severely reduce the unequal wear and thus improve the lifespan of the drive, without a significant increase in the write amplification. In fact, in many cases this mechanism even reduces the write amplification (eliminating the trade-off between low write amplification and a large lifespan altogether).

To assess the performance of this new mechanism we rely both on a mean field model and simulation experiments.

ACM Reference Format:
R. Verschoren and B. Van Houdt, Dept. of Mathematics and Computer Science, University of Antwerp, Belgium. 2019. How to improve the performance of the d-choices garbage collection algorithm in flash-based SSDs. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Flash-based solid state drives (SSDs) contain several flash packages and each such package is organized as an array of blocks. Each block spans a fixed number of pages b and each page is either in the *erase, valid* or *invalid* state. Any I/O operation reads or writes a full page. While read operations are fast and can be performed at all times, a write is more time consuming and can only be performed on a page if the page is in the *erase* state (and changes the state of the page to *valid*). Erase operations occur on a block level, meaning in order to erase a specific page on a block, all the other pages on the same block must be erased as well. As a result SSDs perform out-of-place writes, meaning new data does not overwrite the page where the data was stored (as this would require an erase operation), but is written elsewhere and the SSD controller maintains a table that maps the logical to physical page numbers. At any point in time some of the blocks are marked as special blocks and all new data is written to the special blocks. When a special block runs out of pages in the *erase* state, it becomes a regular block. The garbage collection (GC) algorithm is responsible for creating the special blocks and when invoked the GC algorithm selects a so-called *victim* block. This victim block is erased and becomes a new special block. Any valid pages on the victim block must be written elsewhere before the block can be erased. Hence, the GC algorithm performs internal writes not requested by the host system. As these writes slow down the device, the ratio between the total number of writes and the number of writes requested by the host, termed the *write amplification (WA)*, forms an important SSD performance measure.

A lot of attention has gone to studying the impact of the GC algorithm on the WA [3, 7, 11, 13, 15, 17, 19–21]. In case of uniform random writes the greedy GC algorithm (that always selects the block with the fewest number of valid pages) is optimal [23], but this is no longer the case in general. A popular GC algorithm is the d-choices GC algorithm that selects a block with the fewest number of valid pages among a set of d randomly selected blocks. If some of the logical pages (called the hot pages) are written more frequently than the remaining set of logical pages (called the cold pages), the WA can be reduced significantly by identifying and subsequently separating the cold and hot pages (such that any block contains either hot or cold pages) [5, 6, 10, 16]. This can be achieved using two types of special blocks: a first to support hot page writes and a second to support cold page writes.

Although hot/cold data separation is very effective to reduce the write amplification, blocks that store hot data for a long period of time experience far more erase operations compared to blocks that contain cold data. This is due to the fact that the victim block selected by the GC algorithm is more likely to be a block that stores hot pages as hot pages are invalidated faster. While the number of erase operations that a block can endure may be high in some SSDs, blocks on many contemporary SSDs can only sustain as few as several thousand erase operations [9, 12]. Blocks that are erased too often can no longer guarantee the required data retention times. Thus to improve the endurance of the drive, wear leveling techniques have been developed that aim at reducing the variability in the number of erasures that a block has experienced [4, 8, 14]. These techniques however often have a negative impact on the WA. For instance, the dual pool algorithm of [4] simply exchanges the content of a hot and a cold block at certain times. While such exchange operations tend to level the wear, exchanging the valid

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

pages of two blocks clearly increases the number of internal write operations and thus the write amplification.

In this paper we revisit the setting considered in [21] and [22], where the performance of the d-choices GC algorithm was studied in the presence of hot and cold data. While [21] indicated that the d-choices GC algorithm can achieve a low WA in the presence of hot and cold data; simulation results in [22] showed that the wear can be quite unequal (especially as the hot data gets hotter) and this reduces the lifespan of the drive considerably. To avoid this reduced lifespan, we propose a new mechanism in this paper, called the HCWF(swap) write mode, that results in a much more balanced wear. Furthermore contrary to existing wear leveling algorithms (such as the dual pool algorithm [4]), this mechanism does not result in a significant increase in the WA. In fact in many cases it even slightly reduces the WA. An attractive feature of our mechanism, which also distinguishes it from other existing methods, is that there is no need to maintain erase counters.

The paper is structured as follows. We start by introducing some SSD basics as well as the new HCWF(swap) write mode in Section 2. The mean field model used to assess the WA of this new write mode as well as the validation of this model is presented in Section 3. In Section 4 we use this mean field model to show that this new write mode reduces the WA somewhat in most cases. Simulation results that illustrate large improvements with respect to the unbalanced wear and the lifespan of the drive are presented in Sections 5 and 6, respectively. Conclusions are drawn and future work is discussed in Section 7.

2 SYSTEM OPERATION

2.1 Some SSD basics

As stated before, a flash package is organized as an array of blocks that each span b pages. A page is in the *erase*, *valid* or *invalid* state. At any point in time one or more of the blocks are marked as a special blocks[1], called the write frontiers (WFs). A write to a logical page involves the following 3 steps: (i) the page is written to a page on one of the WFs that is in the *erase* state, thereby changing the state of this page to *valid*, (ii) the map maintained by the controller is updated, (iii) the physical page that was mapped to the logical page before the write occurred is marked as *invalid*.

When the b pages of a WF are full (that is, none are in the *erase* state), the garbage collection (GC) algorithm selects a so-called *victim* block and this block will become a new WF. The valid pages on the victim block are temporarily moved (to RAM) such that the victim block can be erased and are copied back afterwards[2]. If the GC algorithm selected a victim block with j valid pages, j internal write operations are performed on the new WF. These internal writes are then followed by $b-j$ external writes (i.e., writes triggered by the host system) that fill the new WF. As such the write amplification (WA) can be computed as b divided by b minus the mean number of valid pages on a victim block. Hence, in order to achieve a low WA, the mean number of valid pages on the victim block should be low.

To guarantee that a certain fraction S_f of the physical pages is either in the *erase* or *invalid* state at all times, SSDs use over-provisioning. This means that the size of the physical storage space exceeds the size of the logical storage space. If we denote N_{log} as the number of logical blocks on the SSD and $N > N_{log}$ as the number of physical blocks, then $1 - N_{log}/N$ is called the spare factor S_f.

In the next section we discuss the manner in which the WFs are used and selected by the GC algorithm.

2.2 Write modes and victim block selection

We start by discussing the HCWF approach studied in [21, 22], which is closely related to [5] in the sense that it uses two write frontiers: one for the cold and one for the hot pages:

Hot/Cold Write Frontier (HCWF). One block is labeled the hot write frontier (HWF) and another the cold write frontier (CWF) at all times. New hot (cold) data is sequentially written to the HWF (CWF). All the remaining blocks are marked as either hot or cold at all times, depending on whether the block was last used as a HWF or CWF. The initial marking of the blocks (when the drive is empty) is irrelevant. If the HWF becomes full the GC algorithm selects a block, called the victim block. Assume the victim block contains j valid pages, while the CWF has k pages in the *erase* state when the GC algorithm was invoked.

- If the victim block was marked hot, the j valid pages are copied back to the victim block after its pages have been erased and the victim block becomes the new HWF.
- If the victim block was marked cold and $k \geq j$, the j valid pages are copied to the CWF and the victim block becomes the new HWF (which contains b pages in the *erase* state) and is labeled hot. Otherwise if $k < j$, k of the j valid pages of the victim block are copied to the CWF, the remaining $j-k$ pages are written back to the victim block after its pages have been erased and the victim block becomes the new CWF.

In the latter case (i.e., the victim block is marked cold and $k < j$), the GC algorithm is immediately invoked again in search of a new HWF. Finally, when the CWF becomes full, instead of the HWF, the system operates as above if we exchange the terms hot and cold.

A key feature of the HCWF solution is that it dynamically distributes the spare fraction S_f among the hot and cold data partition (formed by the hot/cold block markings) and blocks can move from one partition to the other. As stated in the introduction some blocks may remain hot for long periods of time which hurts the endurance of the drive. To mitigate this we propose the HCWF(swap) write mode. This mode aims at performing occasional swap operations between hot and cold blocks without causing additional internal writes.

Hot/Cold Write Frontier with swap (HCWF(swap)). This write mode operates very similarly to the HCWF mode, except in the following case. Assume the HWF is full, a cold block is the victim and not all the valid pages on the victim block can be copied to the CWF. In case of the HCWF mode the victim becomes the new

[1] A flash package may be composed of multiple dies that each contain several planes, where each plane is organized as an array of blocks that uses its own set of special blocks.

[2] In an actual device the copy operation to RAM is avoided using a single free block. When GC is performed, the victim becomes the new free block and the free block becomes a new WF.

CWF (which involves copying part of the valid pages to the old CWF, erasing the block and rewriting the remaining valid pages) and would trigger the GC algorithm again. Under the HCWF(swap) mode we postpone erasing the victim block and instead select a second victim block, which is forced to be a hot block (see further for details). Thus, we now have two victim blocks: one cold and one hot block. We now remove the valid pages from the cold victim, erase it and copy the valid pages of the hot victim to this block and this (previously cold) victim block becomes the new HWF. Next we erase the hot victim block and copy the valid pages that were stored on the cold victim to this block, which now becomes the new CWF. In this manner a cold block became hot and vice versa without performing an additional costly swap operation. A similar modification is made when the CWF is full, a hot block is the victim and not all the valid pages can be copied to the HWF.

Victim block selection. Whenever the GC algorithm is triggered to select a victim block we make use of the d-choices GC algorithm. Thus the victim block is a block holding the least number of valid pages among a set of d randomly selected blocks. The only exception to this rule is when a second victim block is selected under the HCWF(swap) write mode. In this case, if the first victim block is cold, a hot victim is selected by picking a block with the least number of valid pages among d^* randomly selected hot blocks (and similarly when the roles of hot/cold are reversed).

3 MEAN FIELD MODEL FOR HCWF(SWAP)

3.1 Model description

In this section we adapt the mean field model of [21] used to assess the write amplification of the HCWF write mode, such that it can be used to assess the write amplification of the newly defined HCWF(swap) write mode. As in [7, 20, 21], we consider non-uniform random writes modeled by the hot/cold data model of Rosenblum [18]. In this simple model a fraction f of the logical address space is termed hot and the remaining pages are termed cold, while the logical page numbers of consecutive write requests are independent and the probability that a hot page is requested equals r.

We observe the system consisting of N blocks prior to any call to the GC algorithm and prior to any write request. Let $X_n^N(t) \in S = \{0, \ldots, b\}$ denote the number of valid pages in block $n \in \{1, \ldots, N\}$ and $Y_n^N(t) \in \{h, c\}$ reflect whether this block is labeled hot (h) or cold (c) at the t-th point of observation (i.e., the t-th time the GC algorithm is invoked or a write request is received).

Let $M^N(t)$ be the occupancy measure of $X_n^N(t)$ and $Y_n^N(t)$, that is, $M^N(t) = \{M_{z,i}^N(t) | z \in \{h, c\}, i \in S\}$, while

$$M_{z,i}^N(t) = \frac{1}{N} \sum_{n=1}^{N} 1[X_n^N(t) = i, Y_n^N(t) = z],$$

for $z \in \{h, c\}$ and $i \in S$. In other words, $M_{h,i}^N(t)$ ($M_{c,i}^N(t)$) is the fraction of the total number of blocks N that are labeled hot (cold) and contain i valid pages. To ease the notation we refer to such blocks as type (z, i) blocks.

In [21] $J^N(t) \in \Omega = \{(k, l) | 0 \le k, l \le b\} \backslash \{(b, b)\}$ represented the number of pages written so far in the HWF and CWF. For the HCWF(swap) approach this does not provide us with sufficient information as we need to be able to distinguish between two cases if one of the WFs is full. In the first case a WF just became full (either by an external or internal write) and the victim block will be selected by picking d random blocks. In the second case one of the WFs is full, GC was just triggered, but the victim block was of the other type and not all of the valid pages on the victim block could be moved to the other WF (due to a lack of space). In this case the GC algorithm will be executed again and will select d^* blocks. In order to separate these cases we extend Ω to $\Omega^+ = \Omega \cup \{(b+1, l) | 0 < l < b\} \cup \{(k, b+1) | 0 < k < b\}$, where the state $(b+1, l)$ represents the situation where the HWF is full, l (cold) valid pages for which there was no room left in the CWF remain on the victim block and the GC algorithm will be triggered again to select another victim block by picking d^* hot blocks (i.e., we are in the second case). State $(k, b+1)$ is defined similarly for a full CWF. As a result the GC algorithm is executed at the t-th point of observation if $J^N(t) \in \Omega^+$ is of the form $(b, l), (b+1, l), (k, b)$ or $(k, b+1)$.

It is easy to see that with this change in the range of $J^N(t)$, the process $\{(M^N(t), J^N(t)), t \in \mathbb{N}\}$ is a Markov chain for the HCWF(swap) write mode under the Rosenblum data model. As a direct analysis of this Markov chain appears infeasible, we rely on a mean field model by defining $\bar{M}^N(\tau)$ as the re-scaled process such that $\bar{M}^N(t/N) = M^N(t)$, for $t \in \mathbb{N}$ and $\bar{M}^N(t)$ affine in $[t/N, (t+1)/N]$. Using the mean field framework presented in [1], one can show that the limit process of $(\bar{M}^N(t))$ as N tends to infinity is a deterministic process $\vec{\mu}(t)$, the evolution of which is captured by the set of ODEs given by (1). In other words, for N large and finite t, we can approximate $M^N(t)$ by $\vec{\mu}(t/N)$, which is the unique solution of (1) with $\vec{\mu}(0) = \vec{m}$ (where $M^N(0)$ converges in probability to $\vec{m}$).

The mean field model is defined by means of the deterministic process $\vec{\mu}(t) = \{\mu_{z,i}(t) | z \in \{h, c\}, i \in S\}$, the evolution of which is given by the following set of ODEs:

$$\frac{d\vec{\mu}(t)}{dt} = \vec{F}(\vec{\mu}(t)), \tag{1}$$

with

$$\vec{F}(\vec{m}) = \sum_{(k,l) \in \Omega^+} \pi_{k,l}(\vec{m}) \vec{f}(\vec{m}, k, l)$$

where the drift $\vec{f}(\vec{m}, k, l) = \{f_{(z,i)}(\vec{m}, k, l) | z \in \{h, c\}, i \in S\}$ is defined below and $\vec{\pi}(\vec{m}) = \{\pi_{k,l}(\vec{m}) | (k, l) \in \Omega^+\}$ is the invariant probability vector of $K(\vec{m})$, where $K(\vec{m}) = \lim_{N \to \infty} K^N(\vec{m})$ and $(K^N(\vec{m}))_{i,j} = P[J^N(t+1) = j | J^N(t) = i, M^N(t) = \vec{m}]$, with $i, j \in \Omega^+$. The entries of $K(\vec{m})$ are described in detail further on.

When the GC algorithm is executed it either selects a block with the least number of valid pages among a set of d randomly selected blocks as the victim, or selects a block with the least number of valid pages among a set of d^* blocks that are either all hot or all cold (depending on which WF is full). Given that the GC algorithm is executed while the cooupancy vector equals $\vec{m}$, we denote the probability that the GC algorithm selects a type (z, i) block as $p_{z,i}(\vec{m})$ in the first case and as $q_{z,i}(\vec{m})$ in the latter case when the d^* selected blocks are of type z. Hence we have

$$p_{z,i}(\vec{m}) = \left[\left(\sum_{s=i}^{b} m_s \right)^d - \left(\sum_{s=i+1}^{b} m_s \right)^d \right] \frac{m_{z,i}}{m_i},$$

if $m_i > 0$ (and zero otherwise), where $m_j = m_{h,j} + m_{c,j}$ for $j \in S$, as all the selected blocks must contain at least i valid pages, but not all should contain $i+1$ and ties are broken randomly. Similarly

$$q_{z,i}(\vec{m}) = \left[\left(\sum_{s=i}^{b} m_{z,s}\right)^{d^*} - \left(\sum_{s=i+1}^{b} m_{z,s}\right)^{d^*}\right] \Big/ \left(\sum_{s=0}^{b} m_{z,s}\right)^{d^*}.$$

Note that $\sum_{i=0}^{b} q_{z,i}(\vec{m}) = 1$ for $z = h$ and $z = c$. For further use define $p_{z,i+}(\vec{m}) = \sum_{s>i} p_{z,s}(\vec{m})$.

The drift $f_{(z,i)}(\vec{m},k,l)$ represents the expected change in the number of type (z,i) blocks in between two points of observation given that the occupancy measure equals $\vec{m}$ and the state of the WFs is $(k,l) \in \Omega^+$ at the first point of observation. The drift is identical as in [21] whenever $k, l < b+1$ (and we refer to [21] for a detailed discussion). When $k = b+1$ a type (h,i) block is selected with probability $q_{h,i}(\vec{m})$, which implies that the number of type (h,i) blocks reduces by one and the number of type (h,b) blocks increases by one (which implies no change if the victim block contains only valid pages). We therefore have (where $m_{h,b+1} = m_{c,b+1} = 0$ to ease the notation)

$$f_{h,i}(\vec{m},k,l) = \tag{2}$$
$$\begin{cases} r\frac{(i+1)m_{h,i+1} - i m_{h,i}}{b\rho f} & k < b, l < b, \\ -p_{h,i}(\vec{m}) & i < b, k = b \text{ or } l = b, \\ 1 - p_{h,b}(\vec{m}) - p_{c,(b-l)+}(\vec{m}) & i = b, k = b, \\ p_{h,(b-k)+}(\vec{m}) - p_{h,b}(\vec{m}) & i = b, l = b, \\ -q_{h,i}(\vec{m}) & k = b+1, i < b, \\ 1 - q_{h,b}(\vec{m}) & k = b+1, i = b, \end{cases}$$

and similarly one finds

$$f_{c,i}(\vec{m},k,l) = \tag{3}$$
$$\begin{cases} (1-r)\frac{(i+1)m_{c,i+1} - i m_{c,i}}{b\rho(1-f)} & k < b, l < b, \\ -p_{c,i}(\vec{m}) & i < b, k = b \text{ or } l = b, \\ 1 - p_{c,b}(\vec{m}) - p_{h,(b-k)+}(\vec{m}) & i = b, l = b, \\ p_{c,(b-l)+}(\vec{m}) - p_{c,b}(\vec{m}) & i = b, k = b, \\ -q_{c,i}(\vec{m}) & l = b+1, i < b, \\ 1 - q_{c,b}(\vec{m}) & l = b+1, i = b, \end{cases}$$

with $\rho = 1 - S_f$.

We now discuss the changes required in the transition probability matrix $K(\vec{m})$ compared to [21]. Note that due to the extra states with k or l equal to $b+1$, $K(\vec{m})$ is now a size $b(b+4)-2$ matrix instead of a size $b(b+2)$ matrix. Let $(k,l) \in \Omega^+$ be the state of the WFs at time t and $(k',l') \in \Omega^+$ the state at time $t+1$. The transition probabilities for the HCWF(swap) write mode are given by

$$K(\vec{m})_{(k,l),(k',l')} = \tag{4}$$
$$\begin{cases} r & k < b, k' = k+1, l = l' < b, \\ 1-r & l < b, l' = l+1, k = k' < b, \\ p_{h,k'}(\vec{m}) & k = b, k' \le b, l = l' < b, \\ p_{c,k'}(\vec{m}) & l = b, l' \le b, k = k' < b, \\ p_{c,l'-l}(\vec{m}) & k = b, k' = 0, b \ge l' \ge l, \\ p_{h,k'-k}(\vec{m}) & l = b, l' = 0, b \ge k' \ge k, \\ p_{c,b-l+l'}(\vec{m}) & k = b, k' = b+1, 0 < l' \le l, \\ p_{h,b-k+k'}(\vec{m}) & l = b, l' = b+1, 0 < k' \le k, \\ q_{h,k'}(\vec{m}) & k = b+1, l = l', k' \le b, \\ q_{c,l'}(\vec{m}) & l = b+1, k = k', l' \le b, \\ 0 & otherwise. \end{cases}$$

The first eight cases are as in [21], except that the new state (k',l') in case seven and eight equals $(b+1,l')$ and $(k',b+1)$ respectively, as in these cases a new victim block will be selected next by picking d^* blocks that are all hot (in case seven) or all cold (in case eight). Case nine and ten are new and make use of the probabilities $q_{z,i}(\vec{m})$ as the GC algorithm selects d^* blocks.

3.2 Fixed point computation

While the framework in [1] allows us to show that the sample paths of the sequence of Markov chains converge to the unique solution of the set of ODEs given by (1) over any finite time interval $[0,T]$, this does no necessarily imply that the convergence extends to the stationary regime, unless we can show that the set of ODEs has a fixed point that is a global attractor. Proving global attraction is beyond the scope of the current paper and is still an open problem even for the most basic mean field model used to assess the write amplification in SSDs under uniform random writes [19]. Hence similar to prior work, we compute a fixed point, use this to compute the write amplification and validate its accuracy using simulation.

We determine a fixed point $\vec{v} = \{\nu_{z,i} | z \in \{h,c\}, i \in S\}$ by solving the ODE numerically using Euler's method. While the write amplification WA could be expressed purely in terms of b and the probabilities $p_{z,i}(\vec{v})$ in [21], we now need to take into account that a GC call sometimes samples d random blocks and at other times samples d^* hot (or cold) blocks. Recall that we observe the system just prior to any external write and any GC call. GC calls occur in state $(k,l) \in \Omega^+$ where either k or l is at least b. Hence

$$p_{GC}(\vec{v}) = \sum_{s=0}^{b-1}(\pi_{b,s}(\vec{v}) + \pi_{s,b}(\vec{v})) + \sum_{s=1}^{b-1}(\pi_{b+1,s}(\vec{v}) + \pi_{s,b+1}(\vec{v})),$$

is the probability that we observe the system just prior to a GC call, with $\pi_{k,l}(\vec{v})$ entry (k,l) of the invariant probability vector $\pi(\vec{v})$ of $K(\vec{v})$. Further $p_{GC,d}(\vec{v}) = \sum_{s=0}^{b-1}(\pi_{b,s}(\vec{v}) + \pi_{s,b}(\vec{v}))/p_{GC}(\vec{v})$ is the probability that a GC call samples d blocks, $p_{GC,d^*,h}(\vec{v}) = \sum_{s=1}^{b-1}\pi_{b+1,s}(\vec{v})/p_{GC}(\vec{v})$ the probability that a GC call samples d^* hot blocks and $p_{GC,d^*,c}(\vec{v}) = \sum_{s=1}^{b-1}\pi_{s,b+1}(\vec{v})/p_{GC}(\vec{v})$ the probability that a GC call samples d^* cold blocks. Using these probabilities we can express the write amplification as

$$WA = \frac{b}{b - \sum_{j=0}^{b} j \sum_{z=h,c}(p_{GC,d}(\vec{v})p_{z,j}(\vec{v}) + p_{GC,d^*,z}(\vec{v})q_{z,j}(\vec{v})}, \tag{5}$$

as the sum represents the average number of valid pages on a block selected by the GC algorithm and a new HWF or CWF is only selected when full.

b	S_f	d/d^*	r/f	ODE	simul. (95% conf.)
64	0.15	4/1	0.96/0.24	3.1669	3.1674 ±0.0001
64	0.12	9/10	0.81/0.08	2.5600	2.5604 ±0.0001
64	0.09	12/5	0.94/0.02	1.6543	1.6542 ±0.0001
64	0.06	5/2	0.86/0.13	5.0861	5.0840 ±0.0003
32	0.15	15/40	0.8/0.07	2.1307	2.1312 ±0.0001
32	0.12	50/8	0.77/0.2	3.3725	3.3723 ±0.0002
32	0.09	3/1	0.92/0.12	3.7314	3.7302 ±0.0001
32	0.06	8/15	0.88/0.03	2.5401	2.5399 ±0.0003
16	0.15	4/100	0.8/0.05	1.8939	1.8943 ±0.0001
16	0.12	20/30	0.95/0.15	2.1511	2.1515 ±0.0001
16	0.09	6/3	0.7/0.2	4.2686	4.2670 ±0.0002
16	0.06	10/1	0.9/0.1	3.5805	3.5803 ±0.0002

Table 1: The write amplification for the HCWF(swap) approach: ODE-based results versus simulation experiments for a system with $N\rho = U = 10,000$ blocks for various parameter settings. Relative errors are less than 0.1%.

Euler's method may require several thousand iterations to determine a fixed point. During each iteration the drifts $\vec{f}(\vec{m}, k, l)$ for all $(k, l) \in \Omega^+$ must be computed and we need to determine the steady state probability vector $\pi(\vec{m})$ of $K(\vec{m})$ for some vector $\vec{m}$. Due to the size of the matrix $K(\vec{m})$, each iteration would require $O(b^6)$ time when using standard methods to compute $\pi(\vec{m})$.

To speed up the computation we can proceed in a manner similar to [21] by noting that the drifts given by (2) and (3) do not depend on k, l whenever $k, l < b$. Thus if we denote the drift with $k, l < b$ as $f(\vec{m}, < b)$, then $F(\vec{m})$ can be written as

$$\vec{F}(\vec{m}) = \sum_{(k,l)\in\Omega^+_{\geq b}} \pi_{k,l}(\vec{m})\vec{f}(\vec{m}, k, l) + \left(1 - \sum_{(k,l)\in\Omega^+_{\geq b}} \pi_{k,l}(\vec{m})\right) f(\vec{m}, < b),$$

where $\Omega^+_{\geq b}$ is the subset of Ω^+ containing the $4b - 2$ states with $\min(k, l) \geq b$. Thus, it suffices to compute the $4b - 2$ steady state probabilities $\pi_{k,l}(\vec{m})$ with $(k, l) \in \Omega^+_{\geq b}$. This can be done in $O(b^4)$ time using similar ideas as in [21].

3.3 Model validation

To validate the mean field model, the write amplification for the HCWF(swap) approach computed based on (5) is compared to the write amplification obtained by simulating the Markov chain $\{(X_n^N(t), Y_n^N(t))_{n=1,\ldots,N}, J^N(t)), t \in \mathbb{N}\}$, with 10,000 logical blocks (meaning, $N\rho = 10,000$). Table 1 presents the results for various choices of b, d, d^*, r, f and S_f. It illustrates an excellent agreement between the mean field model and simulation with relative errors below 0.1%. The 95% confidence intervals in Table 1 were computed based on 5 simulation runs, each consisting of 600,000,000 write requests with a warm-up period of 1,000,000 requests.

case	1	2	3	4
$d^* = 1$	2.3626	3.8305	3.7314	3.0869
$d^* = 2$	2.2602	3.5920	3.2453	2.8005
$d^* = 4$	2.1921	3.4329	2.9638	2.6411
$d^* = 8$	2.1553	3.3725	2.8269	2.5680
$d^* = 16$	2.1382	3.3733	2.7663	2.5383
$d^* = 32$	2.1316	3.3932	2.7394	2.5267
$d^* = 64$	2.1299	3.4138	2.7266	2.5219
$d^* = 128$	2.1299	3.4319	2.7202	2.5196

Table 2: The write amplification for the HCWF(swap) approach: impact of d^* for the 4 cases in Table 1 with $b = 32$.

4 WRITE AMPLIFICATION OF HCWF(SWAP)

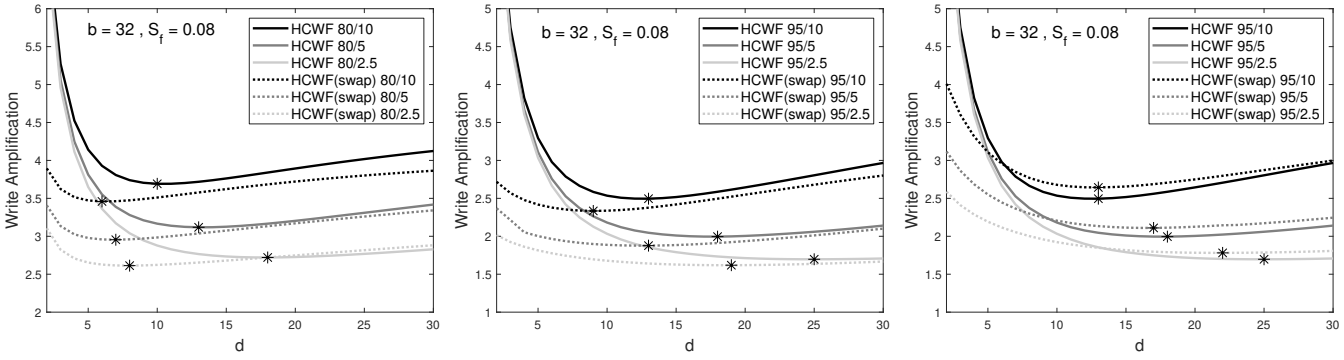
Before we compare the WA of the HCWF(swap) and the HCWF mode, we perform some experiments to shed some light on the impact of d^*. Intuitively it may seem that increasing d^* should always improve performance as we are selecting a victim block among d^* blocks that are all of the same type (being hot or cold). Table 2 shows the impact of d^* on the WA for increasing values of d^* for the four cases with $b = 32$ listed in Table 1. While the WA does decrease as d^* increases in three of the four cases, the second case indicates that this is not necessarily true in all settings. In case 2 of Table 2 the value of d is far from optimal and additional experiments (not shown) suggest that if d is close to optimal, the WA does appear to decrease as d^* increases. For the remainder of this section we set $d^* = 100$ unless otherwise stated, keeping in mind that this may not be optimal.

In Figure 1 we compare the WA of the HCWF and the HCWF(swap) write mode (for $b = 32$ and $S_f = 0.08$ as in [21]). In subfigures 1a and 1b the value of d^* equals 100. In this case we see that the HCWF(swap) write model outperforms the HCWF for nearly any choice of d (except when $r = 0.8$, $f = 0.025$ and d exceeds 15). More importantly, the HCWF(swap) write mode achieves a lower minimum (marked with a star) than the HCWF write mode and the WA of the HCWF(swap) write mode is far less sensitive to the value of d. Even for $d = 2$ or 3 the WA is not too far from optimal, which is in stark contrast to the HCWF write mode.

Subfigure 1c is identical to 1b, except that we lowered d^* to 1 in the HCWF(swap) write mode. We can see that this increases the WA for all d. Nevertheless for small choices of d HCWF(swap) is still superior to HCWF, while for larger d values both write modes perform somewhat similar with the HCWF being the better of the two.

5 PROGRAM-ERASE FAIRNESS OF HCWF(SWAP)

We refer to the number of times that a block has been erased as the number of Program-Erase (PE) cycles that were performed on a block. Let W_{max} reflect the number of PE cycles that a block can tolerate without jeopardizing its required data retention time. In [22] the PE fairness measure was defined as the mean number of PE cycles performed on a block before any block reaches W_{max} PE cycles divided by W_{max}. More formally, if Y_k denotes the number of times the GC algorithm is invoked before any block is erased for



(a) $d^* = 100$, $r = 0.8$ **(b)** $d^* = 100$, $r = 0.95$ **(c)** $d^* = 1$, $r = 0.95$

Figure 1: Write amplification of HCWF versus HCWF(swap) as a function of d for various f and d^* values with $b = 32$ and $S_f = 0.08$.

the k-th time, then the PE fairness is given by

$$PE_f(W_{max}) = \sum_{n \geq 1} P[Y_{W_{max}} = n] \frac{n/N}{W_{max}} = \frac{E[Y_{W_{max}}]}{W_{max} N},$$

as after n GC calls the mean number of PE cycles performed on a block part of a set of N blocks equals n/N. The PE fairness is clearly between 0 and 1 and the main purpose of a wear leveling mechanism is to increase the PE fairness.

The HCWF write mode separates the hot from the cold pages and this significantly reduces the WA [21] as the GC algorithm is more likely to select a victim block containing fewer valid pages. However, if a set of hot logical pages resides a long time on the same set of physical blocks, these blocks endure many PE cycles and this significantly lowers the PE fairness, which in turn hurts the SSD endurance. Simulation experiments in [22] showed that the PE fairness under the HCWF write mode drops sharply as the hot data becomes hotter and values as low as 0.5 for $W_{max} = 5000$ are not uncommon. One approach to improve the PE fairness would be to perform occasional swaps between a hot and a cold block (as proposed in [4]), but this would result in many additional internal writes, which implies an increased WA.

Instead we suggest to use the HCWF(swap) write mode. In the previous section we already indicated that although the HCWF(swap) write mode also occasionally swaps the data between a hot and cold block, it does so without increasing the WA (in fact the WA even slightly decreased in most cases). In Figure 2a and 2b we plot the PE fairness as a function of W_{max} for various combinations of d and d^*. These figures were generated using 50 simulation runs. We note that setting $d^* = 100$ slightly increases the PE fairness compared to setting $d^* = 1$. We also performed additional experiments with other d values and $d^* = 2, 5, 10$ and 30 (not shown) and these suggest that in general the PE fairness increases very slightly as d^* increases. Intuitively this makes sense as we expect that the mean time that a block has remained hot (or cold) slightly increases as the number of valid pages that it contains decreases and it is preferential to perform a swap on a block that has remained hot (or cold) for a longer period of time. Thus, larger d^* values result in a better PE fairness, but the impact of d^* on the PE fairness is very limited. With respect to the impact of d, Figure 2a and 2b may give the impression that smaller d values yield a better PE fairness (as was observed for the HCWF in [22]). While this is true for the most part, the PE fairness decreases again if d is set too small. For instance setting $d = 1$ in Figure 2a and 2b decreases the PE fairness (we left the curves for $d = 1$ out to improve clarity). We further note that achieving a high PE fairness is harder as the hot data becomes hotter and it is therefore expected that the PE fairness is higher for $r = 0.9$ when compared to $r = 0.99$ (with $f = 1 - r$).

To illustrate the effectiveness of the HCWF(swap) write mode we also added the same PE fairness curves for the HCWF write mode in Figure 2c (taken from [22]). Comparing Figures 2a and 2b with Figure 2c clearly indicates that the HCWF(swap) achieves its intended purpose of significantly increasing the PE fairness of the HCWF write mode, especially for larger d values.

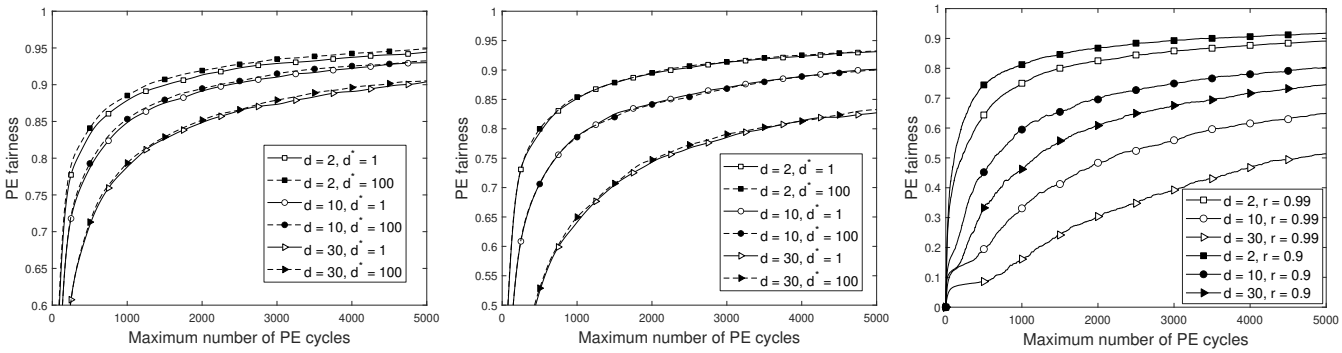
6 ENDURANCE OF HCWF(SWAP)

The main reason for striving for a better PE fairness is to increase the lifespan of the drive. Recall that the lifespan of a drive is determined by the number of PE cycles that a block can endure. If a block is erased too often it can no longer guarantee that its data can be retained for long periods of time. To measure the lifespan, the *SSD endurance (SSD_e)* which equals the expected total number of host writes performed before any block reaches the predefined maximum number W_{max} of PE cycles was proposed in [22]. Hence,

$$SSD_e(W_{max}) = \frac{E[\sum_{j=1}^{Y_{W_{max}}} \sum_{i=0}^{b} (b-i) P[X_j = i]]}{bN},$$

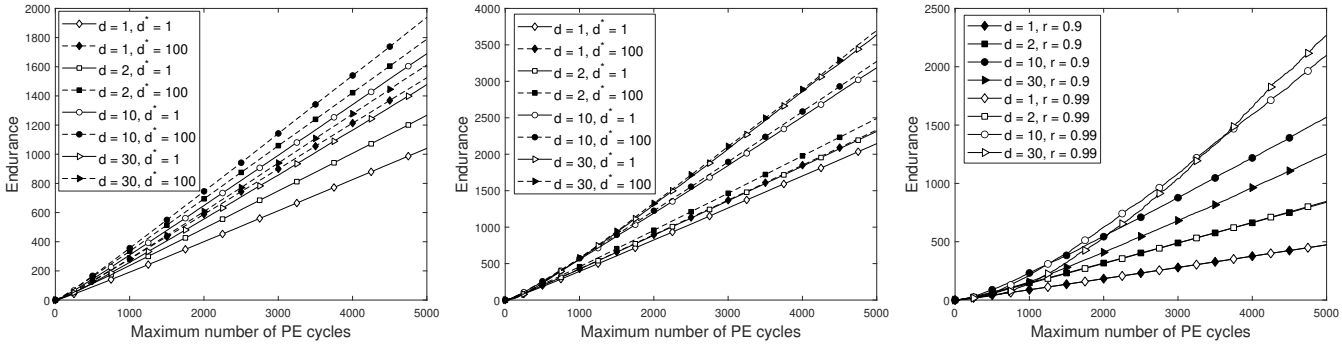
where X_j reflects the number of valid pages on the j-th victim block. Note that the unit used to express the SSD endurance is the total number of Full Drive Writes (FDWs). This endurance definition is the same as the one used in [2] for USB flash drives and in [14] for SSDs.

Figures 3a and 3b show the SSD endurance for various choices of d, d^* and r. Note that despite the fact that the PE fairness for $r = 0.99$ is less than for $r = 0.9$, we see a much higher SSD endurance due to the fact that the WA decreases as the hot data gets hotter. We further note that large d^* values give a better endurance, while the



(a) HCWF(swap) with $r = 0.9$ **(b) HCWF(swap) with $r = 0.99$** **(c) HCWF**

Figure 2: PE fairness as a function of the maximum number of PE cycles with $f = 1 - r$, $b = 32$ and $S_f = 0.10$.



(a) HCWF(swap) with $r = 0.9$ **(b) HCWF(swap) with $r = 0.99$** **(c) HCWF**

Figure 3: SSD endurance as a function of the maximum number of PE cycles with $f = 1 - r$, $b = 32$ and $S_f = 0.10$.

optimal d value depends on the workload and tends to increase with the hotness of the hot data.

More importantly, if we compare the SSD endurance of the HCWF(swap) write mode with the HCWF write mode (depicted in Figure 3c), we see a much higher endurance for the HCWF(swap) write mode. This is especially true for $r = 0.99$, thus as the hot data gets hotter we see a more pronounced improvement. This is in agreement with our expectations as the WA of the HCWF(swap) write mode is never far below that of the HCWF write mode, while it has a much better PE fairness.

For completeness we also included the results for $d = 1$. In case of the HCWF write mode with $d = 1$, all victim blocks are selected uniformly at random and therefore both the fairness and SSD endurance are not influenced by the workload characteristics (i.e., r and f). For the HCWF(swap) mode with $d = d^* = 1$ we do see a better SSD endurance as the GC algorithm is often forced to select a random hot block (which tends to hold fewer hot pages) which reduces the WA and therefore improves the SSD endurance. The hotter the hot data, the more significant the SSD endurance increases.

7 CONCLUSIONS

Wear leveling techniques on flash-based SSDs aim to reduce the variance of the number of erase operations that a block has endured. However, these techniques often require additional internal write operations which further increase the write amplification. In this paper we proposed the HCWF(swap) write mode and showed that it is able to reduce the unequal wear without increasing the write amplification. In fact, the new write mode even improves the write amplification somewhat in many cases.

Regarding the parameters d and d^* of the HCWF(swap) write mode, we showed that setting d^* large and d small offers good performance both in terms of the lifespan of the drive and its write amplification.

The numerical experiments presented in this paper relied on synthetic workloads (of the Rosenblum type). We believe that our insights remain valid for trace-based workloads and plan to look into this next. We also intend to compare the HCWF(swap) write mode with other existing wear leveling mechanisms and to investigate whether the use of erase counters allow any further significant improvements.

ACKNOWLEDGEMENT

This work was supported by the FWO Flanders via research project G024714N entitled *Design and analysis of garbage collection algorithms for flash-based solid state drives.*

REFERENCES

[1] 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.
[2] S. Boboila and P. Desnoyers. Write endurance in flash drives: Measurements and analysis. In *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, FAST'10, pages 9–9, Berkeley, CA, USA, 2010. USENIX Association.
[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] Li-Pin Chang. On efficient wear leveling for large-scale flash-memory storage systems. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, pages 1126–1130, New York, NY, USA, 2007. ACM.
[5] Li-Pin Chang and Tei-Wei Kuo. An adaptive striping architecture for flash memory storage systems of embedded systems. In *Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)*, RTAS '02, pages 187–, Washington, DC, USA, 2002. IEEE Computer Society.
[6] Mei-Ling Chiang, Paul C. H. Lee, and Ruei-Chuan Chang. Using data clustering to improve cleaning performance for flash memory. *Softw. Pract. Exper.*, 29(3):267–290, March 1999.
[7] P. Desnoyers. Analytic models of SSD write performance. *ACM Trans. Storage*, 10(2):8:1–8:25, March 2014.
[8] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37:138–163, 2005.
[9] 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.
[10] J. Hsieh, T. Kuo, and L. Chang. Efficient identification of hot data for flash memory storage systems. *ACM Trans. on Storage*, 2:22–40, 2006.
[11] 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.
[12] Yixin Luo, Yu Cai, Saugata Ghose, Jongmoo Choi, and Onur Mutlu. Warm: Improving nand flash memory lifetime with write-hotness aware retention management. In *MSST*, pages 1–14. IEEE Computer Society, 2015.
[13] 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.
[14] M. Murugan and D. Du. Rejuvenator: A static wear leveling algorithm for NAND flash memory with minimized overhead. In *Proc. of IEEE MSST*, 2011.
[15] S. Odeh and Y. Cassuto. NAND flash architectures reducing write amplification through multi-write codes. In *IEEE 30th Symposium on Mass Storage Systems and Technologies, MSST 2014, Santa Clara, CA, USA, June 2-6, 2014*, pages 1–10, 2014.
[16] D. Park and D. Du. Poster: Hot data identification for flash memory using multiple bloom filters. In *Proc. of USENIX Conference on File and Storage Technologies*, 2011.
[17] 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.
[18] M. Rosenblum and J. K. Ousterhout. The design and implementation of a log-structured file system. *ACM Trans. Comput. Syst.*, 10(1):26–52, February 1992.
[19] 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.
[20] B. Van Houdt. Performance of garbage collection algorithms for flash-based solid state drives with hot/cold data. *Perform. Eval.*, 70(10):692–703, 2013.
[21] B. Van Houdt. On the necessity of hot and cold data identification to reduce the write amplification in flash-based SSDs. *Perform. Eval.*, 82:1 – 14, 2014.
[22] R. Verschoren and B. Van Houdt. On the endurance of the d-choices garbage collection algorithm for flash-based ssds. *ACM Trans. Model. Perform. Eval. Comput. Syst.*, 4(3):13:1–13:23, July 2019.
[23] Y. Yang, V. Misra, and D. Rubenstein. On the optimality of greedy garbage collection for SSDs. *ACM SIGMETRICS Perform. Eval. Rev.*, 43(2):63–65, September 2015.