RZ 3757 (# 99767) 11/23/2009
Computer Science 29 pages

Research Report

Performance Evaluation of the Write Operation In Flash-Based Solid-State Drives

Werner Bux
IBM Research – Zurich, 8803 Rüschlikon, Switzerland
Email: wbu@zurich.ibm.com

Research
Almaden • Austin • Beijing • Delhi • Haifa • T.J. Watson • Tokyo • Zurich

Performance Evaluation of the Write Operation In Flash-Based Solid-State Drives

Werner Bux
IBM Research – Zurich
8803 Rüschlikon, Switzerland

Abstract

We develop an analytical model to describe an important aspect of the operation of a flash-based solid-state drive (SSD), namely, the overhead caused by its out-of-place write and block-erase operation. Depending on the system parameters, in particular the total memory space, the fraction of the memory available for storing user data, and the number of pages per block, the additional write operations performed by the garbage-collection process may significantly reduce the lifetime of a device. We develop a Markov chain model of the SSD operation and use it to explore the performance characteristics of different system designs.

1. Introduction

Flash memory is a non-volatile solid-state memory technology that can be electrically written (programmed), erased, and reprogrammed [2, 9]. Its low power consumption and good shock resistance have made it very popular for portable devices, such as digital cameras, portable music players, mobile phones, and handheld computers. Moreover, the computer industry has recently shown increasing interest in using flash memory as hard-disk replacement in workstations and even enterprise systems. Flash memory does not have the mechanical limitations of hard drives, which makes the idea of a flash-based solid-state drive (SSD) attractive with respect to access time, power consumption, and reliability. What makes the notion of such SSDs less appealing is the cost differential to hard-disk drives, the finite number of erase/write cycles, the fact that flash requires out-of-place writes, and that data has to be erased in large units. There are, however, effective techniques to alleviate these shortcomings, and the current massive industry investment in the development of SSDs suggests that chances are high for flash-based SSDs to become a contender in a sizeable segment of the computer storage space.

In this paper, we address a specific issue of NAND flash-based SSDs: the multiplication effect of write operations that is caused by the fact that flash employs out-of-place write, because in-place updates of individual pages would be prohibitive from a performance point of view. Out-of-place write necessitates a garbage-collection process which reads the valid data from the recycled block and writes them back elsewhere. The multiplication of user writes, often called

“write amplification” [7], is a critical phenomenon in flash-based SSDs, because of its negative effect on their lifetime.

The problem of write costs in flash-based storage has been discussed in the literature, but apart from a few papers, such as [1, 3, 4, 7], mainly in qualitative terms. Our paper describes an analytic approach to model the basic operation of an SSD by a Markov chain to evaluate the performance of the write operation.

2. Basic Operation of Flash-Based SSDs

A NAND flash memory is partitioned into blocks, where each block has a fixed number of pages (typically 64), and each page is of a fixed size (typically 4 KByte). Data are written in a unit of one page, and the erase is performed in a unit of one block. Reads are performed in units of pages. A page can be either programmable or unprogrammable. A programmable page, also called “free page”, becomes unprogrammable once it is written (programmed). A written page contains either valid or invalid data.

In flash-based systems, out-of-place write is used: When a page or a fraction thereof needs to be rewritten, the new data does not overwrite the memory location where the data is currently stored. Instead, the new data is written to another page and the old page is marked invalid. Over time, the SSD accumulates invalid pages and the number of free pages decreases. To make space for new or updated data, invalid pages must be reclaimed. The only way to reclaim a page is to erase the entire block the page pertains to. We consider on-demand reclamation which is triggered when the free space has been completely exhausted. Reclamation or garbage collection happens in multiple steps: First, one of the blocks is selected for reclamation. Second, the valid pages of this block are copied to a reserved free block which thereby becomes part of the device’s actively used storage space. Third, the reclaimed block is erased and becomes the new reserved block.

The effectiveness of the garbage-collection mechanism is influenced by the policy according to which blocks are selected for reclamation. In the main part of this paper we consider the so-called “greedy” policy, in which the block with the smallest number of valid pages is selected for garbage collection. This approach minimizes the number of valid pages that need to be re-written in the course of garbage collection.

For systems with a large number of blocks, implementing the optimal greedy policy may become prohibitive because of the potentially long searches needed to find the optimal block for reclamation. Hence less expensive sub-optimal selection algorithms are of practical interest [7, 8]. To upper-bound the performance of such schemes, we study the performance of a fictitious algorithm

in which the reclaimed block is randomly selected among all blocks having non-zero invalid pages.

Many designs combine reclamation with wear leveling, i.e., blocks are selected for reclamation with the aim of positively affecting the evenness of the wear of the flash cells [4, 5, 10]. Such schemes are beyond the scope of this paper.

3. Quantitative Modeling of the SSD Operation

3.1 Model Assumptions

The memory of the solid state drive we analyze in this paper is assumed to have a total capacity of $t + 1$ blocks; each block consists of c pages. The number of pages available for storing user data is $u \times c$. A total of $(t - u) \times c$ pages is used to temporarily hold pages that have been marked invalid and are waiting for being reclaimed. There is one additional reserved block, which, after garbage collection, will hold the free pages plus copies of any valid pages that were still in the block that was last selected for reclamation. This latter block, at the end of the garbage collection process, will be erased and become the new reserved block. While functionally essential, the reserved block proper does not appear explicitly in our model.

We call the ratio of the storage part actually used for storing user data to the total storage capacity the "utilization" ρ of the SSD:

$$\rho = u / t \,. \tag{1}$$

The discussion and analysis in Sections 3 through 5 are based on the assumption of the greedy reclamation policy. In Section 6, we analyze a scheme in which the reclaimed block is selected randomly from the blocks with non-zero invalid pages. As explained there, this scheme is interesting because it allows one to estimate the maximum performance degradation of sub-optimal selection algorithms.

Write access requests are assumed to be for single pages and to occur independently of each other. We model the system in steady-state, i.e., at a time where the entire memory space available to the user is actually used. In the model, access to a memory location in one of the used pages leads to a switching of the page's state from "valid" to "invalid" and simultaneously switches the state of one of the free pages from "free" to "valid". We assume that the access to each of the valid pages occurs with equal probability $1/(u \times c)$.

3.2 Example

To illustrate the operation and the mathematical description of the system, we consider a small, yet non-trivial system with c = 3 pages per block, t = 6 blocks in total, and the equivalent of u = 4 blocks dedicated to hold user data. (A more general and rigorous description of the process is provided in Sections 4.2 and 4.3.) Figure 1 shows an example of 19 consecutive stages the system steps through. Each stage corresponding to a system state is illustrated by a 3 by 6 matrix filled with the characters v for valid, i for invalid, and space (empty) for free. In each matrix, the left-hand column shows the state of the 3 pages in the first block, the second column the state of the 3 pages in the second block, etc.

Looking at the first stage shown in the figure, we see that the number of v's in the matrix adds up to 12, the product of c = 3 and u = 4. Four pages are marked invalid (i), and 2 are free. Obviously, the number of invalid and free pages has to add up to 6, the product of c = 3 and (t – u) = 2.

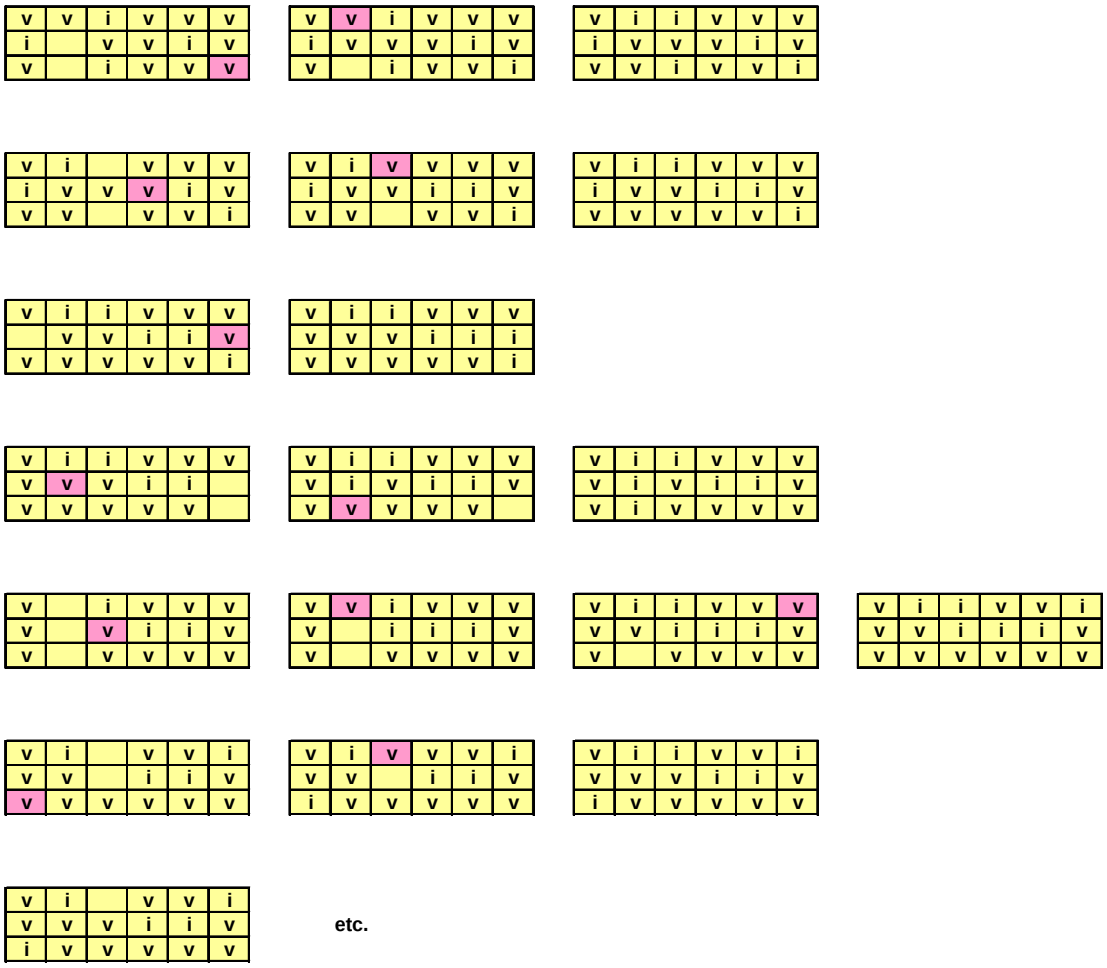

Fig. 1: Write access process and garbage collection in a system with t = 6 blocks, c = 3 pages per block, and the equivalent of u = 4 blocks available for storing user data. Time progresses from left to right and from top to bottom.

Upon arrival of a write request, the system transitions to the next state shown in the second matrix to the immediate right of the first one. The write access is assumed to be to the third page in the 6th block, the matrix element in the bottom right-hand corner. As a consequence of this write access, the targeted page flips its state from valid to invalid, and one of the free pages in the second block changes its state from free to valid.

The next write request happens to hit the first valid page in the second block; in the figure, this is reflected by changing the top element in the second column from v to i and the value of the bottom element in the same column from free to v. After this state change, all free pages are in use, hence the system has to perform garbage collection and reclaim some space for future writes. According to the assumed greedy reclamation strategy, the system selects the block with the smallest number of valid pages for reclamation, which is block #3 in the example. As described in Section 2, the valid pages of the reclaimed block are written to the reserved block, and the selected block #3 is erased. For simplicity, we show the state of the reserved block after reclamation in the column the selected block has "vacated" as a result of garbage collection. In the fourth stage, captured by the leftmost matrix in the second row, there are now 2 free pages in the third block, which corresponds to the two invalid pages in the block that had been selected for reclamation. We call a state of the system immediately prior to the onset of garbage collection a "pre-reclamation state" and a state immediately after termination of garbage collection a "post-reclamation state". The formerly reserved block that after garbage collection contains the free pages and the re-written pages, if any, is called the "write block", as new user data is written into one of its pages until the next garbage-collection epoch.

From the post-reclamation state shown in the leftmost matrix of the second row, the system moves after two writes to the next pre-reclamation state, the right-most matrix in the second row. Here the first block with two valid pages gets selected for reclamation, and so on.

3.3 Optimized State Description

It would be possible to characterize the system states by a $(c \times t)$-tuple in which each element could assume three different values for the corresponding page being in either the valid, invalid, or free state. It is obvious that the resulting state space size of $3^{c \times t}$ would not permit the analysis of systems other than extremely small ones.

Among the conceivable options, we have found the following state description to be optimal in terms of both state space regularity and size.

We describe the state of the system by a vector $(x_0, x_1, \ldots, x_c, y)$ of dimension $(c + 2)$, where the elements x_i denote the number of blocks that momentarily

contain *i* valid or empty pages. The vector element *y* is a number between 1 and *c* that indicates how many valid or free pages the current write block contains.

Before discussing the rationale of this choice, let us apply this description to the example of Fig. 1. The first state in the example (the leftmost matrix in the first row) contains 2 blocks with 3 valid pages each, 2 blocks with 2 valid and 1 invalid page, 1 block with 1 valid and 2 invalid pages, and one block with 1 valid and 2 free pages, i.e., 3 “valid or free” pages. The latter block is the write block; hence *y* is equal to 3. Because of the 2 blocks with the 3 valid pages and the write block with 3 valid or free pages, x_3 equals 3. As there are 2 blocks with 2 valid or free pages, x_2 equals 2. There is one block with one valid or free page, hence $x_1 = 1$, and there is no block with zero valid or free pages, hence $x_0 = 0$. Taken together, this yields the state vector (0,1,2,3,3). As can be deduced from the figure, the second state is described by state vector (0,1,3,2,3). In the second state, one of the 2 valid pages in the write block is hit by the write request, therefore the value of *y* in the subsequent third state is no longer 3 but 2, because the write block no longer contains 3 valid or free pages but only 2. The third state vector becomes (0,1,4,1,2). Transitioning from the pre-reclamation state #3 to the post-reclamation state #4 means that the block with the least number of valid pages (1 in this case) is selected for garbage collection, which means that x_1 is reduced by one, and the new write block contains 3 valid or free pages, i.e., x_3 is increased by 1 and *y* is set to 3. In summary, state #4 is described by the vector (0,0,4,2,3).

The definition of the vector element x_i as the number of blocks having *i* pages that are either valid or free makes it possible to describe the dynamics of the write block analogously to that of the other pages: When one of its valid pages is hit by a write request, the sum of the block’s valid and free pages decreases by one. On the other hand, upon each write access hitting a page in a block that is not the write block, one of the write block’s free pages turns into a valid one, leaving the sum of its valid and free pages unchanged. This homogeneous formal description of all blocks is a major advantage of our state definition.

The transition from a pre- to a post-reclamation state is also rather straightforwardly captured in the state description chosen: As garbage collection produces a new write block with pages that are either free or valid but not invalid, the state vector element with the highest index, x_c, is increased by 1 and *y* assumes a value of *c.* Secondly, the non-zero state vector element with the smallest index is reduced by one because one of these blocks is selected for garbage collection.

Figure 2 shows the state-transition diagram for the example discussed above. The arrows in the figure illustrate the feasible transitions between the states. The states arranged in the same column are characterized by having the same number of valid or free pages:

$$\sigma(x_0, x_1, ..., x_c, y) = \sum_{i=0}^{c} i x_i \ . \qquad (2)$$

Pre-reclamation states are characterized by $\sigma = c \times u = 12$ and appear at the right-hand-side border of the state space. The states at the left-hand-side border have a σ of $c \times (u + 1) = 15$.

As the figure illustrates, the behavior of the system is very regular: Each write access effects a “down-stream” transition to a state with a by-one-smaller σ until the smallest possible σ of $c \times u$ is reached. Upon each down-stream move, the value of y stays the same or is reduced by one.

From a state (x_0, x_1,…., x_q,…, x_c, y) in the pre-reclamation set, the system leaps (“up-stream”) $u - q$ columns to the left where q is the index of the first non-zero element when scanning the state vector from left to right. At the same time, y is set to its maximum value c.

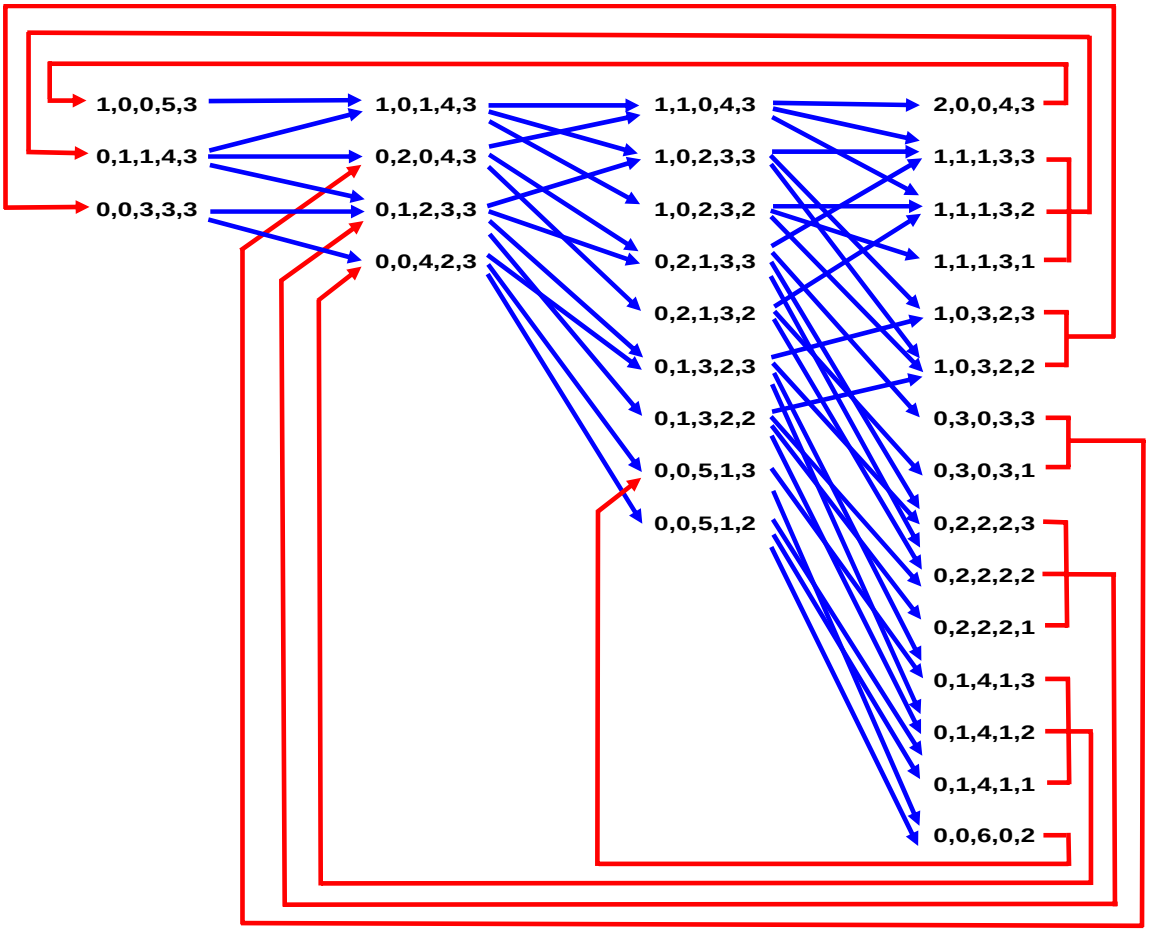

Fig. 2: State-transition diagram of a system with $t = 6$ blocks, $c = 3$ pages per block, and the equivalent of $u = 4$ blocks available for storing user data. For the transition probabilities, see Table 1.

In summary, the behavior of the system is characterized by a series of system changes in the same ("down-stream") direction until the border of the state space is reached, followed by an "up-stream" leap, followed by the next down-stream move, etc. This very regular process is a consequence of our particular state description; it renders the analysis of the system more structured and the numerical computation more transparent and efficient than alternative schemes.

4. Markov Chain Model

4.1 State Definition

We now define the states and behavior of the system considered in more rigorous terms. The assumption of independent single-page access requests makes it possible to describe the system behavior by a homogeneous Markov chain with states

$$S = (X_0, X_1, \ldots, X_c, Y) \tag{3}$$

where the random variable X_i denotes the number of blocks with i valid or empty pages and random variable Y denotes the number of valid or empty pages in the current write block.

The random variables X_0 through X_c and Y can assume non-negative integer values. In order for the vector $(x_0, x_1, \ldots, x_c, y)$ to represent a valid state, the following conditions have to be met:

$$\begin{aligned} &\sum_{i=0}^{c} x_i = t \\ &cu \le \sum_{i=0}^{c} i x_i \le c(u+1) \\ &\min\left\{c, \sum_{i=0}^{c} i x_i + 1 - cu\right\} \le y \le c \end{aligned} \tag{4}$$

4.2 State Transitions

In this section, we derive the one-step transition probabilities of the Markov chain introduced in the preceding section.

4.2.1 State Transitions Caused by Garbage Collection

Under the greedy reclamation policy assumed, the block (or one of the blocks) with the least number of valid pages is selected for reclamation. The valid pages are copied into the reserved block, which then becomes the write block. The selected block is erased and turned into the new reserved block awaiting the next

garbage collection cycle. The garbage collection action is captured in our Markov chain description by having the system change states as described below.

Assume that garbage collection is started when the system is in pre-reclamation state

$$s_{PR} = (x_0, x_1, \ldots, x_{q-1}, x_q, x_{q+1}, \ldots, x_c, y)$$
$$\text{with} \quad x_0 = x_1 = \ldots = x_{q-1} = 0;\ x_q > 0;\ \sum_{i=1}^{c} i \cdot x_i = c \cdot u;\ 1 \le y \le c \quad . \tag{5}$$

Because the block (or one of the blocks) with the least number of valid pages is selected for reclamation, the non-zero state variable x_q with the smallest index q is decreased by 1 and the variable x_c with the highest index is increased by 1. The latter reflects the fact that the block that was turned from the reserved into a utilized block contains only free or valid pages; in other words, a block (the new write block) with c valid or free pages has been added to the system. Consequently, the variable y in the state description assumes the value c, as it has to point to one of the blocks with c valid or free pages. In summary this yields, for $k \in \{1, \ldots, c\}$, the following expression for the transition probabilities between pre-and post-reclamation states:

$$p((x_0, \ldots, x_{q-1}, x_q, x_{q+1}, \ldots, x_c, y), (x_0, \ldots, x_{q-1}, x_q - 1, x_{q+1}, \ldots, x_c + 1, c)) =$$

$$= \begin{cases} 1 & \text{if} \quad x_0 = \ldots = x_{q-1} = 0, \quad x_q > 0, \quad \sum_{i=0}^{c} i x_i = cu, \quad 1 \le y \le c \\ 0 & \text{otherwise} \end{cases} \quad . \tag{6}$$

As part of the garbage collection process, q valid pages of the selected block are being re-written into the hitherto reserved block, the new write block.

4.2.2 State Transitions Caused by Write Requests

As mentioned in Section 3.1, it is assumed that a write-access request targets any of the valid pages independently and with equal probability. To derive the state transition probabilities, assume that the updated page is part of a block with k valid or free pages. Under this assumption, the value of the state variable x_k is reduced by 1 and the value of variable x_{k-1} is increased by 1. As captured by Eq. (7), we have (for each value of k between 1 and c) to differentiate between three cases:

(***A***) The write access hits a page with the same number of valid pages as the write page, but not the write page itself. This implies $k = y$, and occurs when one of the $(x_k - 1) \times k$ pages in non-write blocks with k valid pages is hit, which happens with probability $(x_k - 1) \times k/cu$, see the first line on the right-hand side

of Eq. (7). In this case, the number of valid or free pages in the write block does not change, hence $y' = y$.

(***B***) The write access hits a block with a different number of valid pages than the write block, i.e., $k \neq y$. This happens when one of the $x_k \times k$ pages in blocks with k valid pages is hit. The transition probability is given in the second line of Eq. (7). In this case, the number of valid or free pages in the write block does not change, hence $y' = y$.

(***C***) The write access hits a valid page of the write block, i.e., $k = y$ and $y' = y - 1$. The number of valid pages in the write block is equal to the total number k of valid or free pages in the write block minus the number of free pages. The latter is equal to the total number of valid or free pages in the state considered minus the total number of valid pages. This translates into the transition probability given in the third line of Eq. (7).

For all other combinations of k, $x_0,\ldots, x_c$, y, and y', the transition probability is zero.

$$p((x_0, x_1,..., x_k,..., x_c, y),(x_0, x_1,..., x_{k-1}+1, x_k-1,..., x_c, y')) \quad =$$

$$= \begin{cases} \dfrac{(x_k-1)k}{cu} & if \quad k = y, \quad y' = y, \quad x_k > 0 \\ \dfrac{x_k k}{cu} & if \quad k \neq y, \quad y' = y, \quad x_k > 0 \\ \dfrac{k - \sum_{i=1}^{c} i x_i + cu}{cu} & if \quad k = y, \quad y' = y-1, \quad x_k > 0 \\ 0 & \text{otherwise} \end{cases} \qquad (7)$$

For the example underlying the state-transition diagram shown in Fig. 2, Table 1 provides the transition probabilities that were calculated using Eqs. (6) and (7).

Table 1: State-transition probabilities of a system with t = 6 blocks, c = 3 pages per block, and the equivalent of u = 4 blocks available for storing user data. For an illustration of the state space, see Fig. 2.

Current State Vector	Next State Vector	Trans. Prob.	Current State Vector	Next State Vector	Trans. Prob.	Current State Vector	Next State Vector	Trans. Prob.
1,0,0,5,3	1,0,1,4,3	12/12	1,1,0,4,3	2,0,0,4,3	1/12	2,0,0,4,3	1,0,0,5,3	1
0,1,1,4,3	1,0,1,4,3	1/12		1,1,1,3,3	9/12	1,1,1,3,3	0,1,1,4,3	1
	0,2,0,4,3	2/12		1,1,1,3,2	2/12	1,1,1,3,2	0,1,1,4,3	1
	0,1,2,3,3	9/12	1,0,2,3,3	1,1,1,3,3	4/12	1,1,1,3,1	0,1,1,4,3	1
0,0,3,3,3	0,1,2,3,3	6/12		1,0,3,2,3	6/12	0,3,0,3,3	0,2,0,4,3	1
	0,0,4,2,3	6/12		1,0,3,2,2	2/12	0,3,0,3,1	0,2,0,4,3	1
1,0,1,4,3	1,1,0,4,3	2/12	1,0,2,3,2	1,1,1,3,2	2/12	1,0,3,2,3	0,0,3,3,3	1
	1,0,2,3,3	9/12		1,1,1,3,1	1/12	1,0,3,2,2	0,0,3,3,3	1
	1,0,2,3,2	1/12		1,0,3,2,2	9/12	0,2,2,2,3	0,1,2,3,3	1
0,2,0,4,3	1,1,0,4,3	2/12	0,2,1,3,3	1,1,1,3,3	2/12	0,2,2,2,2	0,1,2,3,3	1
	0,2,1,3,3	9/12		0,3,0,3,3	2/12	0,2,2,2,1	0,1,2,3,3	1
	0,2,1,3,2	1/12		0,2,2,2,3	6/12	0,1,4,1,3	0,0,4,2,3	1
0,1,2,3,3	1,0,2,3,3	1/12		0,2,2,2,2	2/12	0,1,4,1,2	0,0,4,2,3	1
	0,2,1,3,3	4/12	0,2,1,3,2	1,1,1,3,2	2/12	0,1,4,1,1	0,0,4,2,3	1
	0,1,3,2,3	6/12		0,3,0,3,1	1/12	0,0,6,0,2	0,0,5,1,3	1
	0,1,3,2,2	1/12		0,2,2,2,2	9/12			
0,0,4,2,3	0,1,3,2,3	8/12	0,1,3,2,3	1,0,3,2,3	1/12			
	0,0,5,1,3	3/12		0,2,2,2,3	6/12			
	0,0,5,1,2	1/12		0,1,4,1,3	3/12			
				0,1,4,1,2	2/12			
			0,1,3,2,2	1,0,3,2,2	1/12			
				0,2,2,2,2	4/12			
				0,2,2,2,1	1/12			
				0,1,4,1,2	6/12			
			0,0,5,1,3	0,1,4,1,3	10/12			
				0,0,6,0,2	2/12			
			0,0,5,1,2	0,1,4,1,2	8/12			
				0,1,4,1,1	1/12			
				0,0,6,0,2	3/12			

4.3 State Probabilities

To determine the write amplification, we need to know the probability distribution of the number of valid pages in the blocks selected for reclamation. To determine this distribution, it is sufficient to observe the system only when it is in a pre-reclamation state, determine the transition probabilities p_{ij}^* between successive pre-reclamation states i and j, and from those calculate the steady-state probabilities π_i.

To proceed along this line, we group the states into subsets with the same number of free pages n or, equivalently, the same σ as defined in Eq. (2). The

pre-reclamation states (n = 0) carry the numbers 1 through a_0. States having n free pages ($n \in \{1,\ldots, c\}$) are numbered from $a_{n-1} + 1$ through a_n.

The probability p_{ij}^* that the system transitions in one or multiple steps from any of its states with number i ($i \in \{1, 2,\ldots, a_c\}$) to the next pre-reclamation state with number j ($j \in \{1, 2,\ldots, a_0\}$) can be determined in a step-by-step fashion:

Step 1: for $i \in \{a_0 + 1,..., a_1\}$:

$$p_{ij}^* = p_{ij} \tag{8a}$$

(The one-step transition probabilities p_{ik} are given by Eq. (7).)

Steps 2 to c: for $n \in \{2,..., c\};\ i \in \{a_{n-1} + 1,..., a_n\}$

$$p_{ij}^* = \sum_{k=a_{n-1}+1}^{a_n} p_{ik} \cdot p_{kj}^* \tag{8b}$$

(The one-step transition probabilities p_{ik} are given by Eq. (7).)

Step c+1: for $i \in \{1,..., a_0\}$

$$p_{ij}^* = \sum_{k=a_0+1}^{a_c} p_{ik} \cdot p_{kj}^* \tag{8c}$$

(The one-step transition probabilities p_{ik} are given by Eq. (6).)

If for $i, j \in \{1,\ldots, a_0\}$, we denote by

$$P^* = [p_{ij}^*] \tag{9}$$

the ($a_0 \times a_0$) transition probability matrix and by

$$\pi = (\pi_1, \pi_2,..., \pi_{a_0}) \tag{10}$$

the state probability vector of the Markov chain, then it holds for the steady-state probabilities of the pre-reclamation states [6] that

$$\pi = \pi \cdot P^*$$

$$\pi \cdot e = 1 \quad , \tag{11}$$

where e is the ($a_0 \times 1$) column vector with all elements equal to one. Solving this

system of linear equations finally yields the state probabilities needed for calculating the write amplification as discussed in the next section.

4.4 Write Amplification

In the context of this paper, write amplification *A* is defined as the average of the actual number of (system) page writes per user page write.

To determine *A*, we observe the system at garbage-collection epochs, i.e., when the system is in a pre-reclamation state s_{PR} as defined in (5). As explained in Section 4.2.1, garbage collection in this state involves the re-writing of *q* pages. As this happens with probability π_i, the expected number of rewritten pages is given by

$$\overline{q} = \sum_{i=1}^{a_0} \pi_i q_i \,. \tag{12}$$

In a complete cycle consisting of all write accesses between two subsequent garbage-collection epochs plus one garbage-collection action, exactly *c* pages get either written or rewritten, $c - \overline{q}$ pages of these are user page writes. Hence write amplification *A* is given by

$$A = \frac{c}{c - \sum_{i=1}^{a_0} \pi_i q_i} \,, \tag{13}$$

where the probabilities π_i are the solutions of the system of linear equations (11).

4.5 State Space Size

The number of linear equations in (11) can be reduced by a factor close to 1/*c* without loss of generality or exactness of the solution. As illustrated in Fig. 2 and formally captured in Eq. (6), all pre-reclamation states (x_0, x_1,…, x_c, *y*) with the same values of their first *c* + 1 elements x_0,…, x_c have one and only one transition to the same post-reclamation state. Replacing them by a single “macro pre-reclamation state” (x_0,…, x_c) will therefore not alter the behavior of the system. When garbage collection is performed in states with identical values of x_0,…, x_c, the same number of pages is re-written (same value of q_i in Eq. (13)). Hence knowledge of the *macro* pre-reclamation state probabilities is fully sufficient for computing the write amplification.

In Appendix A1, we prove that the number of macro pre-reclamation states can be recursively computed using the function *N*(*m*, *n*, *s*) defined as follows: For *m* = 1 and *n* = 1 and non-negative integers *s*, *N* is defined by

$$N(1,n,s)=\begin{cases}1 & if\ s\le n\\ 0 & if\ s>n\end{cases}$$

$$N(m,1,s)=\begin{cases}1 & if\ s\le m\\ 0 & if\ s>m\end{cases}, \tag{14}$$

respectively, and for integers $m, n \geq 2$ and non-negative integers s by

$$N(m,n,s)=\begin{cases}N(m,n-1,s) & if\ s<n\\ N(m,n-1,s)+N(m-1,n,s-n) & if\ s\ge n\end{cases}. \tag{15}$$

For a system with c pages per block, a total number of t blocks, and u blocks available for storing user data, the number of macro pre-reclamation states is given by

$$G(c,t,u)=N(c,t,cu)\ . \tag{16}$$

Table 2 provides numerical values for $G(c,t,u)$ for a wide range of parameter combinations.

Table 2: Number of macro pre-reclamation states $G(c, t, u)$ for different numbers of pages per block c, total number of blocks t, and number of utilized blocks u.

t	4	16	64	256
c = 4				
u = 1/4 t	5	64	2,280	123,464
u = 1/2 t	8	177	8,173	479,837
u = 3/4 t	5	64	2,280	123,464
c = 8				
u = 1/4 t	15	2,755	6,208,394	> 1E+9
u = 1/2 t	33	17,575	83,916,031	> 1E+9
u = 3/4 t	15	2,755	6,208,394	> 1E+9
c = 16				
u = 1/4 t	64	487,842	> 1E+9	> 1E+9
u = 1/2 t	177	8,908,546	> 1E+9	> 1E+9
u = 3/4 t	64	487,842	> 1E+9	> 1E+9
c = 32				
u = 1/4 t	351	363,829,479	> 1E+9	> 1E+9
u = 1/2 t	1,143	> 1E+9	> 1E+9	> 1E+9
u = 3/4 t	351	363,829,479	> 1E+9	> 1E+9
c = 64				
u = 1/4 t	2,280	> 1E+9	> 1E+9	> 1E+9
u = 1/2 t	8,173	> 1E+9	> 1E+9	> 1E+9
u = 3/4 t	2,280	> 1E+9	> 1E+9	> 1E+9

4.6 An Upper Bound on the Write Amplification

Because the numerical evaluation of our solution is non-trivial and confined to moderate-size systems, a simple upper bound on the write amplification is of some practical value. In addition, it sheds interesting theoretical light onto the behavior of our system.

The approach to upper-bound write amplification A is to upper-bound in Eq. (13) the term

$$\sum_{i=1}^{a_0} \pi_i q_i \ .$$

This is accomplished by finding conditions among the system parameters c, t, and u (or $\rho = u/t$) such that for any of the pre-reclamation states, the number of rewritten pages q_i in (13) is no larger than a given value $k < c$.

In Appendix A2, we prove that for $k \in \{0, 1,\ldots,c-1\}$, the following bound on the write amplification holds:

$$A \le \hat{A} = \frac{c}{c-k} \qquad if \quad \rho < \hat{\rho} = \frac{k+1}{c} \tag{17}$$

5. Approximate Model

It is possible to reduce the complexity of the model by making the heuristic assumption that in any system state the write block is always among the blocks with the highest momentary number of valid or empty pages. Under this assumption, the random variable Y in the state description (3) can be omitted. This simplification leads to a considerable reduction of the compute resources required (processing time and memory) compared with the exact solution. Comprehensive tests showed that the error in the write amplification caused by this approximation is generally below 2%.

6. Random Reclaimed-Block Selection

In practice, the optimal “greedy” policy, which selects the block having the least valid pages for reclamation, can become computationally expensive when the number of blocks is very high. It has been suggested to use less-expensive sub-optimal selection algorithms, for example to search for the optimal block only within an appropriate subset (“window”) of the blocks [7, 8].

Rather than studying any particular such policy, we will subsequently analyze a fictitious scheme in which the reclaimed block is randomly selected from among all blocks having non-zero invalid pages. The rationale for investigating this case

is that one can assume that all “intelligent” policies will yield a better write amplification than the random scheme.

Compared with the case of the greedy policy studied so far, the analysis of the random selection scheme differs in the state transitions resulting from the garbage collection process. Under the greedy policy, there is one and only one transition from any pre-reclamation state, namely, the one that meets the optimal selection criterion of the greedy policy. This fact is reflected in Eq. (6) and illustrated in Fig. 2. In contrast, under the random reclaimed-block selection policy, the system can, with certain probabilities that depend on the pre-reclamation state vector, transition to multiple different post-reclamation states. In our analysis, this change is reflected by replacing Eq. (6) by the following formula, which holds for $r \in \{0, 1, \ldots, c-1\}$:

$$p((x_0,...,x_r,...,x_c,y),(x_0,...,x_r-1,...,x_c+1,c)) = \tag{18}$$

$$= \begin{cases} \dfrac{x_r}{\sum_{k=0}^{c-1} x_k} & if \quad x_r > 0, \quad \sum_{k=0}^{c} k x_k = cu, \quad 1 \le y \le c \\ 0 & \text{otherwise} \end{cases}$$

A transition from pre-reclamation state # i ($i \in \{1, \ldots, a_0\}$) with state vector $(x_0, \ldots, x_r, \ldots, x_c, y)$ to post-reclamation state $(x_0, \ldots, x_r - 1, \ldots, x_c + 1, c)$ is associated with the re-writing of r valid pages. Hence, the expected number of re-written pages when garbage collection is performed in state # i is given by

$$\overline{r_i} = \frac{\sum_{r=0}^{c-1} r \cdot x_r}{\sum_{k=0}^{c-1} x_k} = \frac{\sum_{r=0}^{c-1} r \cdot x_r}{c \cdot u - x_c} \ . \tag{19}$$

In analogy to Eq. (13), we obtain the write amplification for the random reclaimed-block selection as

$$\overline{A} = \frac{c}{c - \sum_{i=1}^{a_0} \overline{\pi}_i \overline{r}_i} \ , \tag{20}$$

where $\overline{\pi}_i$ is the steady-state probability of pre-reclamation state # i ($i \in \{1, \ldots, a_0\}$). The state probabilities are computed by solving the system of equations (11), in which the transition probabilities are determined on the basis of Eqs. (7) and (18) rather than (7) and (6).

7. Numerical Results

In a practical SSD, the number of pages per block is 64 and the total number of blocks is on the order of 1,000 to 10,000. Despite our optimized state definition, the size of the state space for these parameter values is several orders of magnitude beyond the computation capabilities of today's computers. Fortunately, it turns out that the analysis of systems with smaller parameter values still yields very useful insights into the general behavior of this type of storage systems and that it is possible to heuristically extrapolate the measures of interest to the parameter ranges of practical significance.

Figure 3 shows the typical write amplification characteristic of the system considered as a function of the storage utilization ρ, the ratio of the space available for storing active (i.e., valid) user data to the total storage space of the device. Shown in the chart are curves for 5 different values of c, the number of pages per block. All curves exhibit a very moderate write amplification effect up to about 60% utilization. As the utilization gets into the 70 to 90% range, the overhead caused by the out-of-place writes in combination with the block-based erase operation becomes substantial. Figure 3 also shows that write amplification becomes worse for larger numbers of pages per block. We will get back to this effect in the context of Figs. 5 to 8.

Figure 4 repeats the results for c = 10 from Fig. 3 and puts them into relation with the upper bound of Eq. (16). It can be seen that the bound is not very tight. However, as it follows nicely the real $A(\rho)$ characteristic and is trivial to compute, it can serve as a guideline for a first-order estimation of the write amplification for parameter ranges that are impossible to address with the exact solution.

Figure 5 plots the write amplification A as a function of the total number of blocks t. For each of the 5 curves, the utilization, i.e., the ratio of u and t, is kept constant. We notice again the strong dependency of A on ρ. It is interesting to observe that for fixed values of ρ, A quickly approaches a horizontal asymptote. The reason for this effect can be understood by inspecting the state probabilities π_i of the pre-reclamation states. It can be generally observed that the probabilities of having to re-write 0, 1, 2,… pages during garbage collection change only minimally when t grows beyond 50. In other words, larger values of t and u result in larger state spaces, but the write amplification is only minimally affected if the ratio of these parameters is kept constant.

The individual results marked with different symbols (see the legend) and connected by solid lines represent numerical results generated by our model. The smooth lines are used for ease of representation and should not be read as suggesting that the corresponding values are continuous. The dashed lines represent heuristic extrapolations of the computed results, indicating the trend for growing values of t. Given the benign behavior of the $A(t)$ characteristic described above, this "engineering guess" should not be too far off the truth.

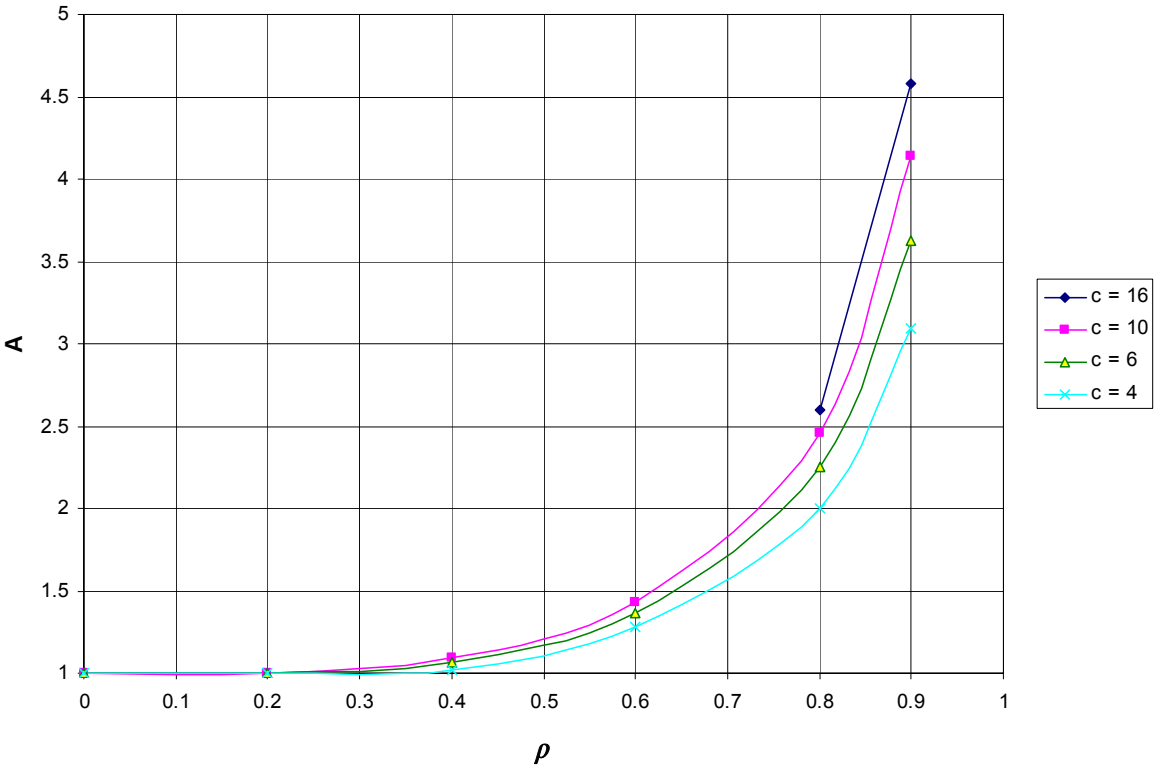

Fig. 3: Write amplification A as function of the utilization ρ for t = 10 blocks and different numbers of pages per block c.

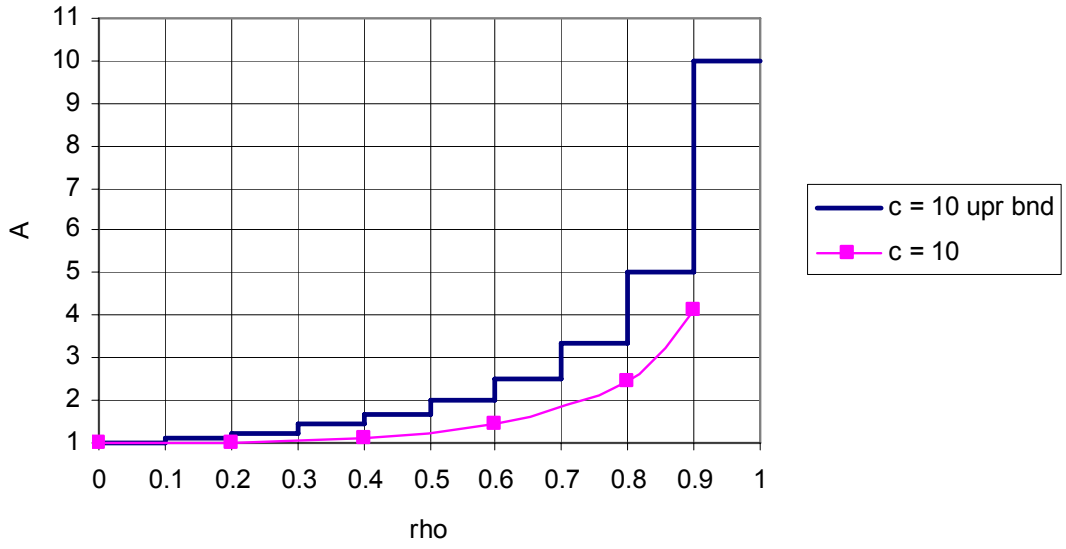

Fig. 4: Write amplification A as function of the utilization ρ for t = 10 blocks and c = 10 pages per block. Exact solution and upper bound.

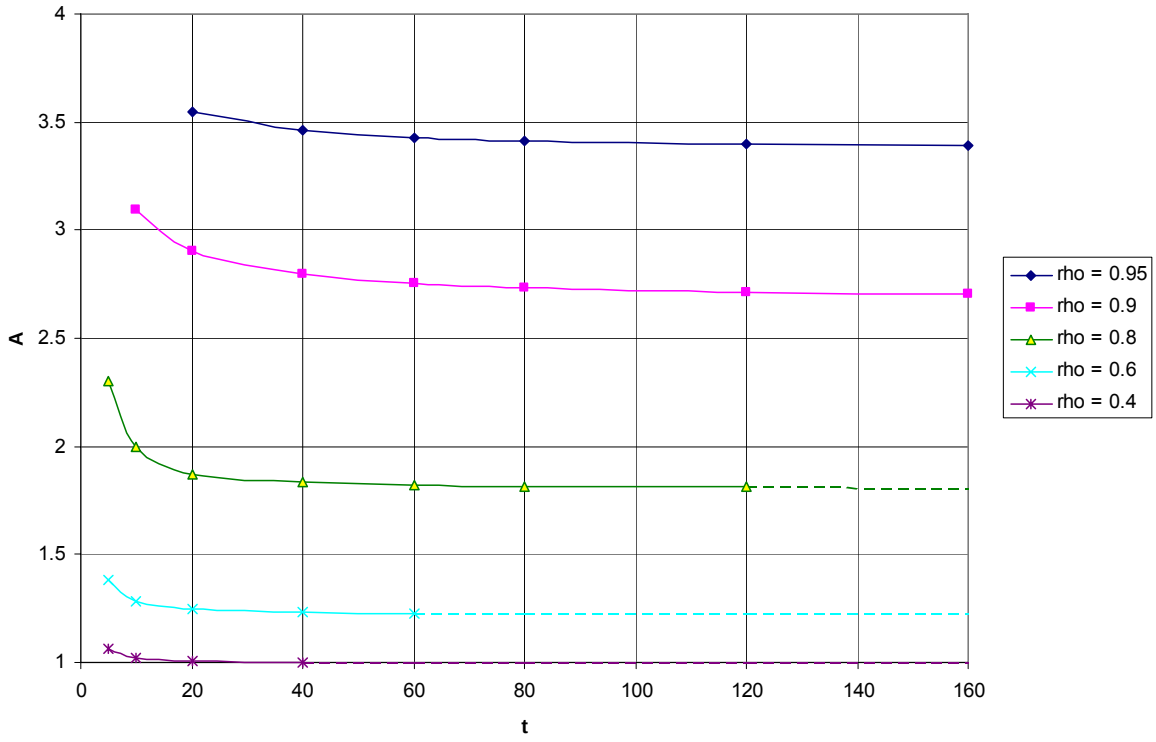

Fig. 5: Write amplification *A* as function of the total numbers of blocks *t* for different values of the utilization ρ (0.95, 0.9, 0.8, 0.6, 0.4). *c* = 4 pages per block.

The intention behind Figs 6 to 9 is to provide a reasonably complete picture of the parameter space that can be addressed by our model. For different fixed values of ρ, the figures show the write amplification *A* as a function of the total number of blocks *t*. The four charts demonstrate that *A* decreases with growing storage size, but levels off at values of *t* below 100. The explanation of this effect is essentially the same as for Fig. 5.

As can be seen from the four figures, the write amplification is better for smaller values of *c* because smaller blocks reduce the effect of the block erase operation. This effect is particularly strong at higher utilizations, whereas at lower utilizations, the advantage of a smaller *c* is less pronounced, unless one would consider extremely small number of pages per block. These results could be interpreted as suggesting the use of small numbers of pages per block. One needs to keep in mind though that optimizing *A* with respect to *c* is only one, and likely not the most relevant, consideration: Because erases are performed in units of blocks and take a comparatively long time, employing a small number of pages per block would be very inefficient.

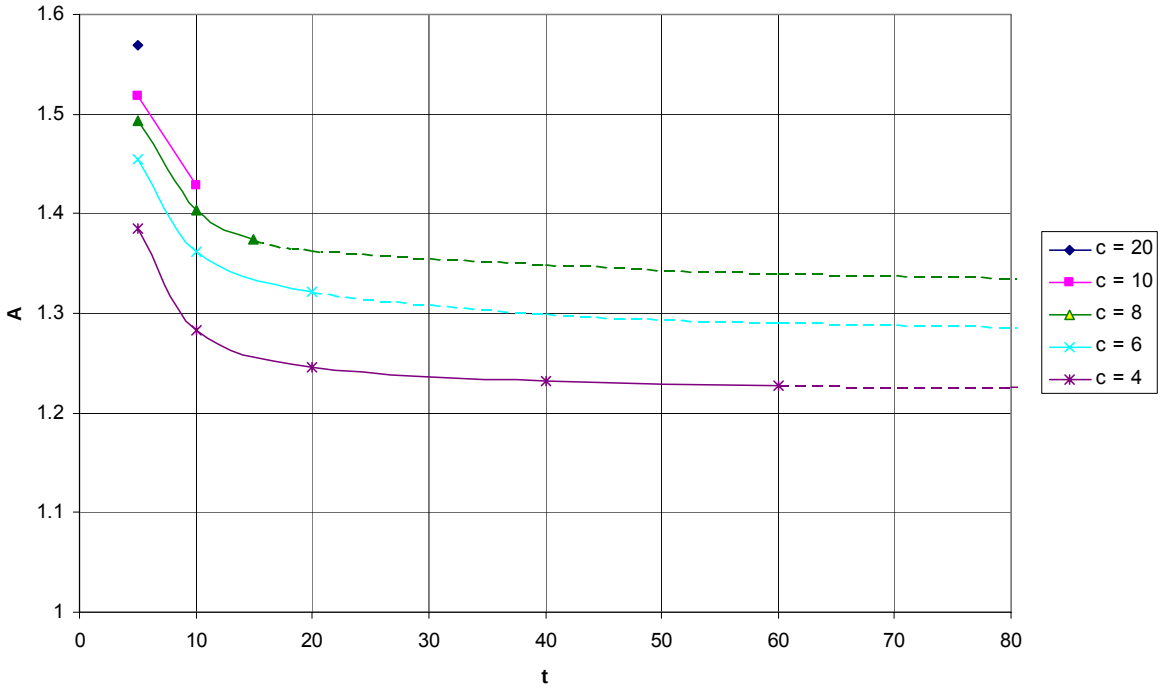

Fig. 6: Write amplification *A* as function of the total number of blocks *t* for different values of the number of pages per block *c*. Utilization ρ = 0.6.

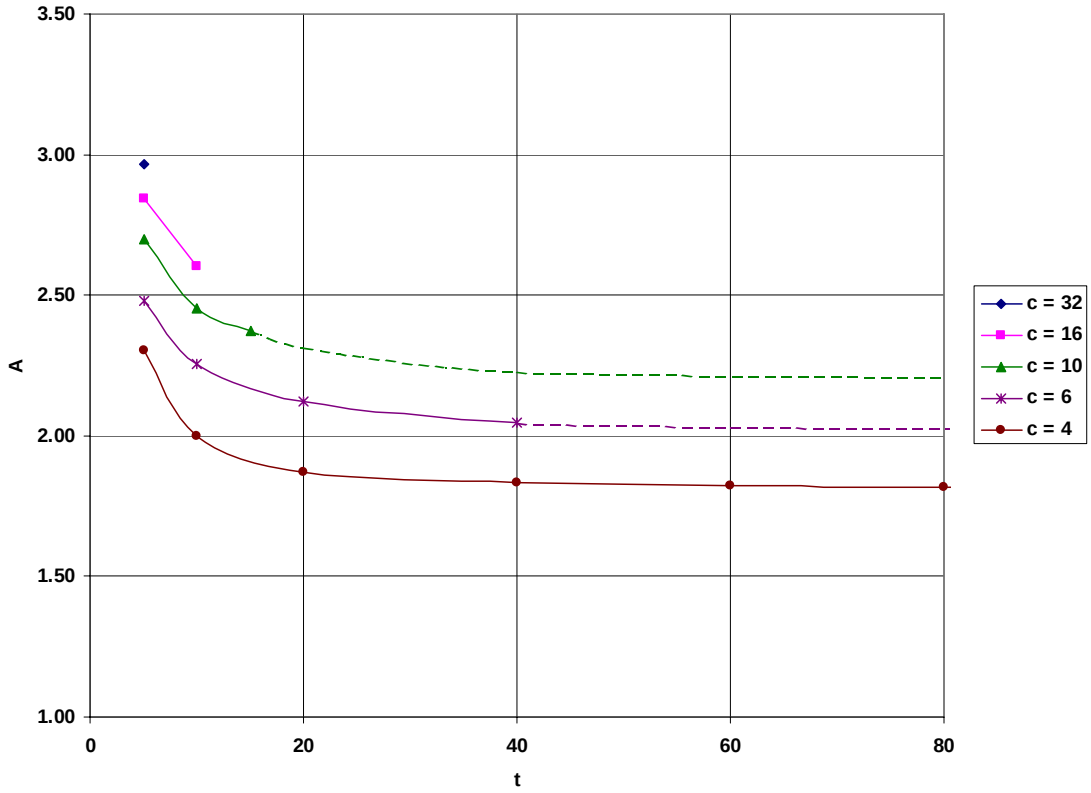

Fig. 7: Write amplification *A* as function of the total number of blocks *t* for different values of the number of pages per block *c*. Utilization ρ = 0.8.

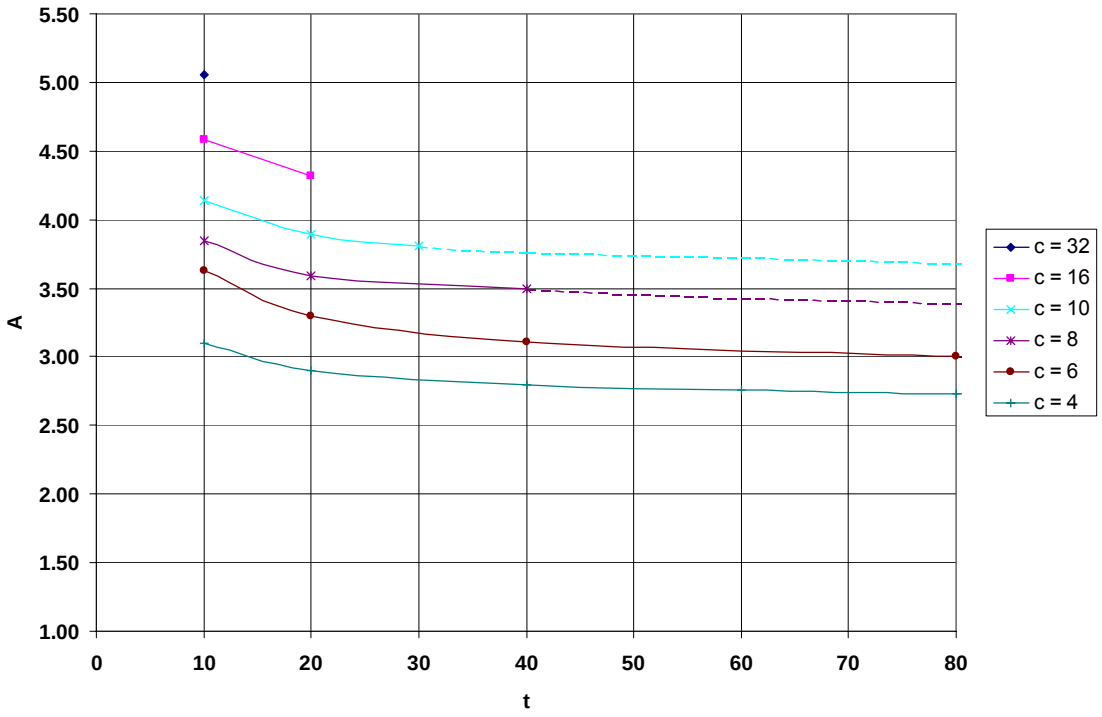

Fig. 8: Write amplification *A* as function of the total number of blocks *t* for different values of the number of pages per block *c*. Utilization ρ = 0.9.

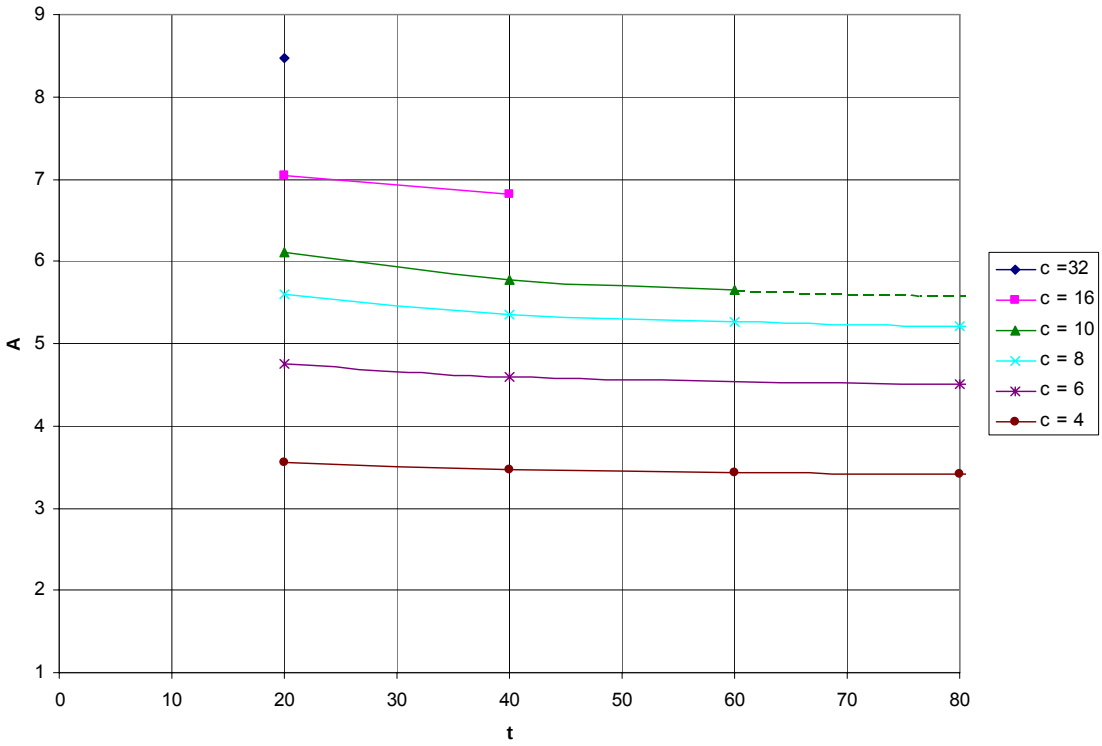

Fig. 9: Write amplification *A* as function of the total number of blocks *t* for different values of the number of pages per block *c*. Utilization ρ = 0.95.

We are able to compute at least a few cases for c = 32, but the state space for the value of practical interest, c = 64, is just slightly too big for our analysis program. As described in Sections 3.3 and 4, we have taken great care to come up with an efficient state definition and recursive computation of the transition probabilities of the Markov chain, so a fundamentally more efficient exact analysis of the problem is hard to imagine. It is conceivable, though, that the implementation of our analytic approach can be further optimized with respect to both computation time and memory requirements. This will be the subject of future work. A complementary approach also worth pursuing is to develop an approximate solution that works for larger parameter ranges than the exact model described in this paper.

Figures 10 and 11 address the dependency of A on c in a different form that allows us to extrapolate the write amplification characteristics to relatively large values of c. Figure 10 shows the write amplification A as a function of the number of pages per block c for different values of the total number of blocks t at a utilization of ρ = 0.6. Because the curves level off for values of c > 30, heuristically extrapolating the curves up to c = 64 appears to be justified. Furthermore, the write amplification exhibits little increase as the total number of blocks increases above 40. It is therefore pretty safe to estimate the write amplification for c = 64 and large numbers of blocks to be around 1.46. This is in agreement with the simulation results in [7].

The same considerations hold for the results shown in Fig. 11 for a utilization of ρ = 0.8. For this utilization, the extrapolation of our results suggest a write amplification of roughly 2.3 for c = 64 and large numbers of blocks.

Finally, we direct our attention to the “random” reclaimed-block selection scheme discussed in Section 6. For c = 4 and t = 10, Fig. 12 shows the write amplification for the greedy and the random scheme. It can be seen that the absolute and relative differences between the two policies are minimum at low and high utilizations. One way of looking at the results is to compare the “admissible” utilizations, i.e., the values of ρ that yield the same write amplification for the two schemes. The horizontal distance between the two curves visualizes the gain in terms of better use of the storage, or equivalently, in terms of storage size savings achieved by putting intelligence into the selection algorithm. As the figure shows, this gain is substantial, e.g., a factor of 2 in storage size for a write amplification of approx. 1.3. Figure 13 shows the corresponding results for c = 10.

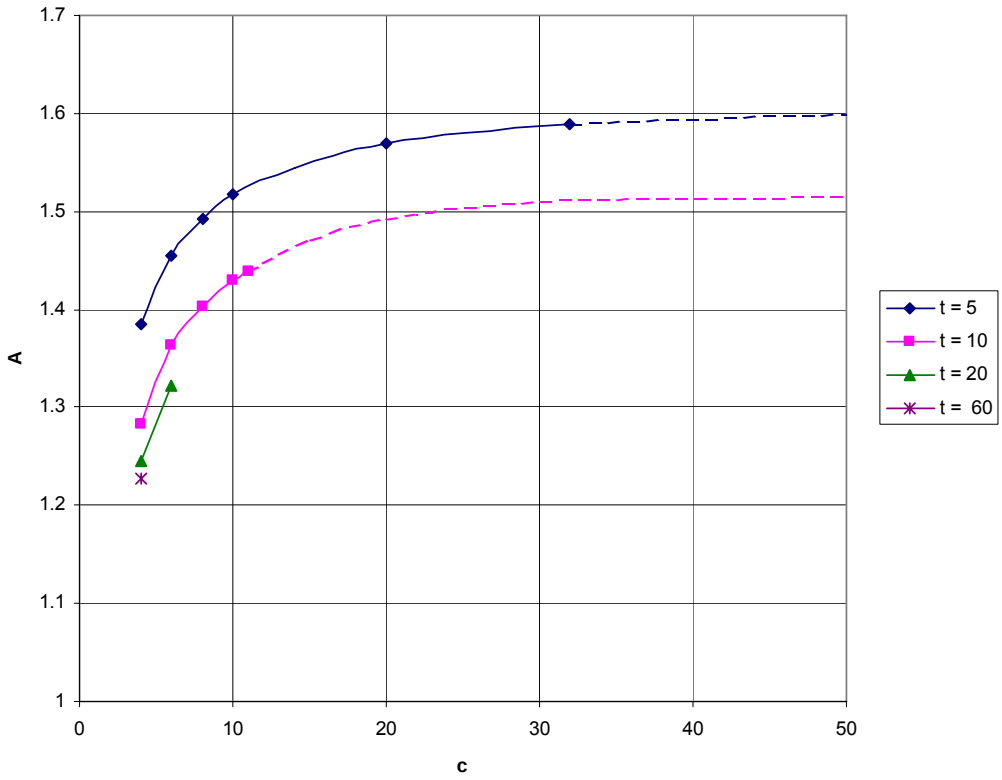

Fig. 10: Write amplification *A* as function of the number of pages per block *c* for different values of the total number of blocks *t*. Utilization ρ = 0.6.

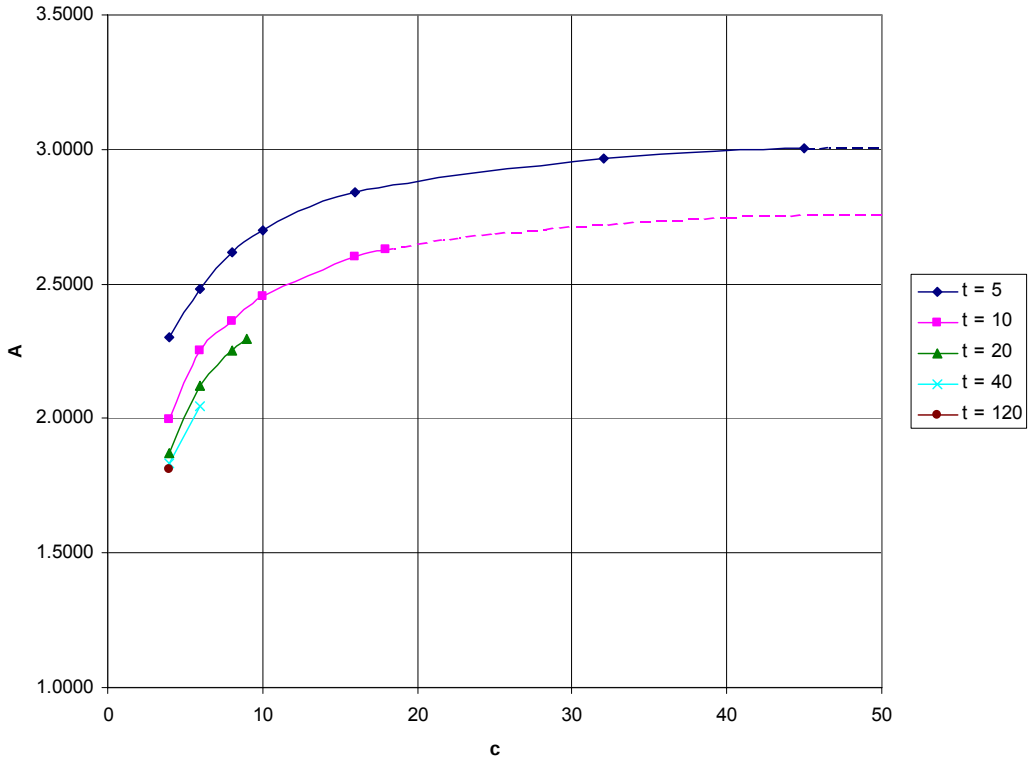

Fig. 11: Write amplification *A* as function of the number of pages per block *c* for different values of the total number of blocks *t*. Utilization ρ = 0.8.

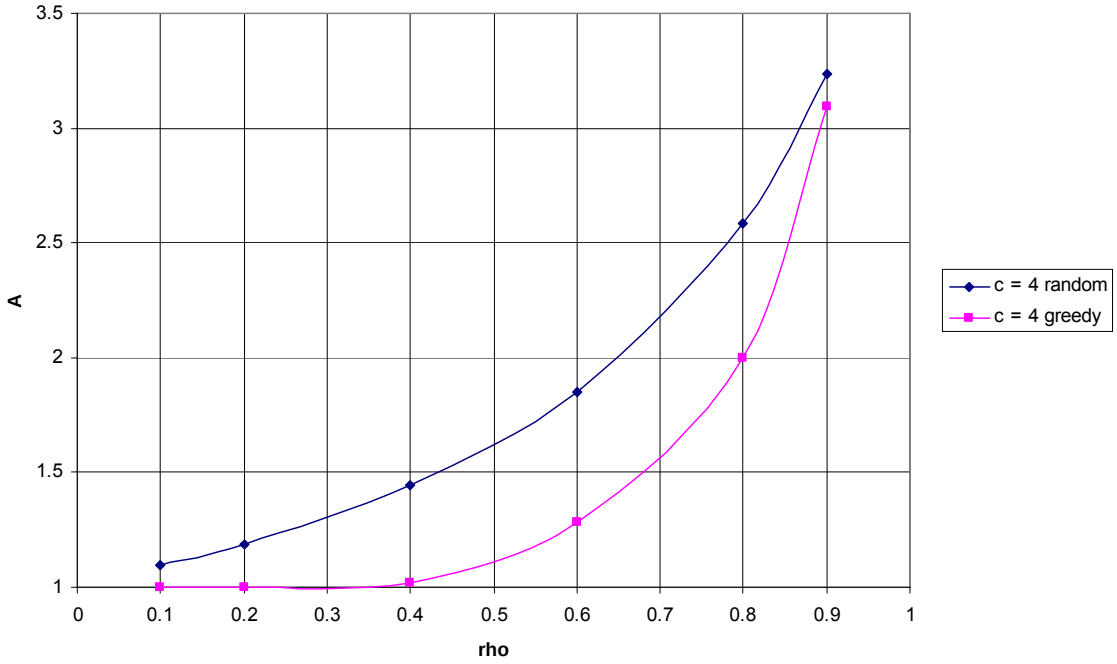

Fig. 12: Write amplification *A* as function of the utilization ρ for t = 10 blocks and c = 4 pages per block for the greedy and the random reclaimed-block selection scheme.

7. Conclusions

In this paper, we developed an analytical model to describe the basic operation of a flash-based SSD with the aim of quantifying the cost of its out-of-place write and block-erase operations. Depending on the system parameters, e.g., the total storage space, the fraction of the storage available for storing user data, and the number of pages per block, write amplification, i.e., the ratio of system writes to user writes, may significantly affect the lifetime of a device. A Markov chain model of the SSD operation made it possible to study the sensitivity of the write efficiency to the above parameters. The state description of the Markov chain was optimized for minimum size and complexity of the state space, which made it possible to perform an exact analysis of moderate-size systems.

Through an upper-bound result and heuristic extrapolation, we have been able to estimate the write amplification beyond the limits of the exact solution. Furthermore, we performed an analysis of a fictitious scheme, in which the reclaimed blocks are selected randomly from the blocks with non-zero invalid pages. This analysis provides a limit on the performance degradation resulting from a sub-optimal reclamation policy.

Acknowledgment

The work described in this paper has greatly benefited from being part of a broader research project in which a flexible flash-based solid state drive system has been architected, designed, and prototyped. The author is indebted to his colleagues involved in this effort, in particular Roy Cideciyan, Evangelos Eleftheriou, Robert Haas, Xiao-Yu Hu, Ilias Iliadis, Peter Müller, and Roman Pletka, for numerous insightful discussions. He is grateful to Ilias Iliadis for providing a sample of simulation results to cross-check the validity of the analysis. Last but not least, he would like to thank Christoph Hagleitner, Haris Pozidis, and Martin Schmatz for their help in establishing the compute environment for this work.

Appendices

A1: Size of the State Space

In this appendix, we derive a recursive formula for computing the number of macro pre-reclamation states and upper- and lower-bounds for the number of all states.

For non-negative integers $x_0,\ldots, x_n$ and s, and for positive integers m and n, let $M(m, n, s)$ be the set of vectors defined by

$$M(m,n,s) = \{(x_o, x_1,..., x_m) \mid \sum_{i=0}^{m} x_i = n, \sum_{i=0}^{m} i x_i = s\} \quad . \tag{A.1}$$

We split $M(m, n, s)$ into 2 disjoint sets $M_0(m, n, s)$ and $M_1(m, n, s)$ as follows:

$$\begin{aligned} M_0(m,n,s) &= \{(x_o, x_1,..., x_m) \mid (x_o, x_1,..., x_m) \in M(m,n,s) \wedge x_0 = 0\} \\ M_1(m,n,s) &= \{(x_o, x_1,..., x_m) \mid (x_o, x_1,..., x_m) \in M(m,n,s) \wedge x_0 > 0\} \quad . \end{aligned} \tag{A.2}$$

Next, let us map each vector $(x_0,\ldots, x_m) \in M_0$ onto a vector $(x'_0,\ldots, x'_{m-1})$ by setting

$$x'_i = x_{i+1} \quad \text{for } i \in \{0,1,...,m-1\} \quad . \tag{A.3}$$

Note that

$$\sum_{i=0}^{m-1} x'_i = n \quad \text{and} \quad \sum_{i=0}^{m-1} i \cdot x'_i = s - n \quad . \tag{A.4}$$

Therefore the set of vectors $(x'_0,\ldots, x'_{m-1})$ is equal to $M(m-1, n, s-n)$ if $m \geq 2$ and $s \geq n$.

Secondly, we map each vector $(x_0,\ldots,x_m) \in M_1$ onto a vector $(x''_0,\ldots,x''_m)$ by setting

$$\begin{aligned} x''_0 &= x_0 - 1 \\ x''_i &= x_i \qquad for\ i \in \{1,...,m\} \end{aligned} \quad . \tag{A.5}$$

If $n \geq 2$, the set of vectors $(x''_0,\ldots,x''_m))$ is equal to $M(m, n-1, s)$ because

$$\sum_{i=0}^{m} x''_i = n-1 \quad \text{and} \quad \sum_{i=0}^{m} i \cdot x''_i = s \quad . \tag{A.6}$$

Because each element of $M_0(m, n, s)$ is mapped exactly onto one element of $M(m-1, n, s-n)$ and each element of $M_1(m, n, s)$ exactly onto one element of $M(m, n-1, s)$, and since

$$\begin{aligned} M_0(m,n,s) \cup M_1(m,n,s) &= M(m,n,s) \\ M_0(m,n,s) \cap M_1(m,n,s) &= \emptyset \end{aligned} \quad , \tag{A.7}$$

we obtain for $m, n \geq 2$ the following recursive relationship for the number of elements $N(m, n, s)$ of $M(m, n, s)$:

$$N(m,n,s) = \begin{cases} N(m,n-1,s) & if\ s < n \\ N(m,n-1,s) + N(m-1,n,s-n) & if\ s \geq n \end{cases} \quad . \tag{A.8}$$

The initial values of $N(m, n, s)$ for $m = 1$ and for $n = 1$ are given by

$$N(1,n,s) = \begin{cases} 1 & if\ s \leq n \\ 0 & if\ s > n \end{cases} \tag{A.9}$$

$$N(m,1,s) = \begin{cases} 1 & if\ s \leq m \\ 0 & if\ s > m \end{cases} .$$

The above equations lend themselves to a straightforward recursive computation of $N(m, n, s)$.

Macro pre-reclamation states $(x_0,\ldots, x_c)$ for a system with c pages per block, t blocks in total, and u blocks available for user data are characterized by

$$\sum_{i=0}^{c} x_i = t \quad and \quad \sum_{i=0}^{c} i \cdot x_i = cu \quad . \tag{A.10}$$

Therefore the number of macro pre-reclamation states is given by

$$G(c,t,u) = N(c,t,cu) \ . \tag{A.11}$$

For the purpose of counting the total number of states, we group all valid states (x_0, x_1,…, x_c, y) with the same values for x_0, x_1,…, x_c into a single so-called macro state (x_0,…, x_c). From the definitions (3) and (4) of the system states and the above definition of $N(m, n, s)$, it can be seen that for a system with c pages per block, a total number of t blocks, and u blocks available for storing user data, the number of macro states is given by

$$Q(c,t,u) = \sum_{s=cu}^{c(u+1)} N(c,t,s) \ . \tag{A.12}$$

Determining the number of all ("micro-") states as defined in (3) and (4), is possible but quite involved and will be omitted in this report. However, it is relatively straightforward to upper- and lower-bound it as follows:

Let $n(c, t, s)$ be the number of (micro-) states (x_0,…, x_c, y) as defined in (3) and (4) with

$$s = \sum_{i=0}^{c} i x_i \ . \tag{A.13}$$

The total number of all (micro-) states is then given by

$$q(c,t,u) = \sum_{s=cu}^{c(u+1)} n(c,t,s) \ . \tag{A.14}$$

For a given value of s, the number of different values that the state vector element y can assume is upper-bounded by

$$k(c,t,s) = \begin{cases} c\cdot(u+1)-s & for\ s \in \{c\cdot u, c\cdot u+1, ..., c\cdot(u+1)-2\} \\ 1 & for\ \ s \in \{c\cdot(u+1)-1, c\cdot(u+1)\} \end{cases} \ . \tag{A.15}$$

This implies

$$n(c,t,s) \le k(c,t,s)\cdot N(c,t,s) \quad for\ s \in \{cu, ..., c(u+1)\} \ . \tag{A.16}$$

Therefore the number of all states is bounded by

$$Q(c,t,u) \le q(c,t,u) \le \sum_{s=cu}^{c(u+1)} k(c,t,s)\cdot N(c,t,s) \ . \tag{A.17}$$

A.2: Upper Bound on the Write Amplification

In this appendix, we derive an upper bound on the write amplification *A* for the greedy reclamation policy. We proceed by finding a sufficient condition on the system parameters *c, t*, and *u* such that

$$x_k > 0 \qquad if \quad x_0 = ... = x_{k-1} = 0 \quad for \quad k \in \{0,1,...,c-1\} \tag{A.18}$$

when the system is in a pre-reclamation state

$$\begin{aligned} & s_{PR} = (x_0, x_1, \ldots, x_k, \ldots, x_c, y) \\ & with \quad x_0 = x_1 = \ldots = x_{k-1} = 0;\ x_k > 0;\ \sum_{i=1}^{c} i \cdot x_i = c \cdot u;\ 1 \le y \le c \end{aligned} \quad . \tag{A.19}$$

For any $k \in \{0, 1, \ldots, c-1\}$, it follows from (A.18) and (A.19) that

$$x_k = t - \sum_{i=k+1}^{c} x_i \tag{A.20}$$

and

$$\sum_{i=k+1}^{c} x_i = c \cdot u - k \cdot t - \sum_{i=k+1}^{c} (i-k-1) \cdot x_i \quad . \tag{A.21}$$

Inserting (A.21) into (A.20) yields

$$x_k = (k+1) \cdot t - c \cdot u + \sum_{i=k+1}^{c} (i-k-1) \cdot x_i \quad . \tag{A.22}$$

To enforce $x_k > 0$, it is sufficient that

$$(k+1) \cdot t - c \cdot u > 0 \tag{A.23}$$

since $\sum_{i=k+1}^{c} (i-k-1) \cdot x_i \ge 0$.

If condition (A.23) is met for all pre-reclamation states, then $x_k > 0$ when all x_i with index $i < k$ are equal to zero. This implies that there will never be more than *k* valid pages that need to be rewritten during garbage collection. In other words, the term

$$\sum_{i=1}^{a_0} \pi_i q_i$$

in (12) is upper-bounded by *k*, and it follows from (12) and (A.23) that *A* is bounded by

$$A \le \hat{A} = \frac{c}{c-k} \qquad if \quad \rho < \hat{\rho} = \frac{k+1}{c}; \quad k \in \{0,1,...,c-1\} \quad . \tag{A.24}$$

References

[1] Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. Design tradeoffs for SSD performance. *Proceedings of the USENIX 2008 Annual Technical Conference*, pp. 57-70, June 2008.

[2] Joe Brewer and Manzur Gill (eds.). *Nonvolatile Memory Technologies with Emphasis on Flash: A Comprehensive Guide to Understanding and Using Flash Memory Devices*, Wiley-IEEE Press, 2008.

[3] Li-Pin Chang, Tei-Wei Kuo, and Shi-Wu Lo. Real-time garbage collection for flash-memory storage systems of real-time embedded systems, *ACM Transactions on Embedded Computing Systems*, Vol. 3, No. 4, June 2004, pp. 1-26.

[4] Li-Pin Chang. On efficient wear leveling for large-scale flash-memory storage systems, *Proceedings of the 2007 ACM Symposium on Applied Computing*, Seoul, Korea, pp. 1126-1130

[5] Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories, *ACM Computing Surveys*, Vol. 37, No. 2, June 2005, pp. 138-163.

[6] Donald Gross and Carl M. Harris. *Fundamentals of Queueing Theory*, Second Edition, John Wiley & Sons, New York, 1985.

[7] Xiao-Yu Hu, Evangelos Eleftheriou, Robert Haas, Ilias Iliadis, and Roman Pletka. Write amplification analysis in flash-based solid state drives, *Proceedings of The Israeli Experimental Systems Conference "SYSTOR 2009,"* Haifa, Israel, Article no. 10, ACM, May 2009.

[8] Jai Menon and Larry Stockmeyer. An age-threshold algorithm for garbage collection in log-structured arrays and file systems, in J. Schaeffler, editor, *High Performance Computing Systems and Applications*, Kluwer Academic Publishers, pp. 119–132, 1998.

[9] Micron Technical Note NAND Flash 101: An Introduction to NAND Flash and How to Design It into Your Next Product, 2006.

[10] M. Wu and Willy Zwaenepoel. eNVy: A non-volatile, main memory storage system. *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 86-97, 1994.