Queueing Syst (2014) 77:115–148
DOI 10.1007/s11134-014-9405-y

Stochastic modeling and optimization of garbage collection algorithms in solid-state drive systems

Yongkun Li · Patrick P. C. Lee · John C. S. Lui

Received: 29 April 2013 / Revised: 23 October 2013 / Published online: 5 April 2014

Abstract Markov chains and mean-field analysis are powerful tools and widely used for performance analysis in large-scale computer and communication systems. In this paper, we consider the application of Markov modeling and mean-field analysis to solid-state drives (SSDs). SSDs are now widely deployed in mobiles, desktops, and data centers due to their high I/O performance and low energy consumption. In particular, we focus on characterizing the performance–durability tradeoff of garbage collection (GC) algorithms in SSDs. Specifically, we first develop a stochastic Markov chain model to capture the I/O dynamics of large-scale SSDs, then adapt mean-field analysis to derive the asymptotic steady state, based on which we are able to easily analyze the performance–durability tradeoff of a large family of GC algorithms. We further prove the model convergence and generalize the model for all types of workload. Inspired by this model, we also propose a randomized greedy algorithm (RGA) which has a single tunable parameter to trade between performance and durability. Using trace-driven simulation on DiskSim with SSD add-ons, we demonstrate how RGA can be parameterized to realize the performance–durability tradeoff.

An earlier conference version of the paper appeared in ACM SIGMETRICS 2013 [36]. In this journal version, we present all proofs, revise the presentation of mean-field model, analyze write amplification, and provide more rigorous arguments for our discussion throughout the paper.

Y. Li (✉)
School of Computer Science and Technology, University of Science and Technology of China,
Hefei, China
e-mail: yongkunlee@gmail.com

Y. Li · P. P. C. Lee · J. C. S. Lui
The Chinese University of Hong Kong, New Territories, Hong Kong
e-mail: pclee@cse.cuhk.edu.hk
J. C. S. Lui
e-mail: cslui@cse.cuhk.edu.hk

Keywords Solid-state drives · Garbage collection · Wear-leveling · Cleaning cost · Markov model · Mean-field analysis

Mathematics Subject Classification 60H10 · 60K30 · 68M20

1 Introduction

Markov chains and mean-field analysis are powerful tools in analyzing the performance of computer and communication systems. In this paper, we focus on the application of Markov chain modeling to analyze the performance of the newly emerging storage devices, solid-state drives (SSDs). In particular, we show that the underlying Markov chain model is effective to characterize the system state of SSDs which consist of a large number of storage blocks. Based on the derived system state, one can easily analyze the performance–durability tradeoff of garbage collection (GC) algorithms in SSDs. We first develop a stochastic Markov chain model to capture the I/O dynamics of large-scale SSDs, then apply mean-field analysis to derive the asymptotic steady state, and finally analyze the performance–durability tradeoff of GC algorithms based on the derived system state. To elaborate on our work, we first present the necessary background on SSDs, then briefly illustrate the performance–durability tradeoff of GC algorithms, and finally state our contributions as well as the overall structure of this paper.

1.1 Background on SSDs

The increasing adoption of SSDs in mobile devices to large-scale search engines is revolutionizing the way we process data. Today's SSDs are mainly built on NAND flash memory, and provide number of attractive features, i.e., high performance in I/O throughput, low energy consumption, and high reliability due to their shock resistance property. As the SSD price per gigabyte decreases [21], not only desktops are replacing traditional hard-disk drives (HDDs) with SSDs, but there is a growing trend toward using SSDs in data centers [19,27].

SSDs have inherently different I/O characteristics from traditional HDDs. An SSD is organized in *blocks*, each of which usually contains 64/128/256 *pages* that are typically of size 4/8 KB each [1,13,40]. It supports three basic operations: *read*, *write*, and *erase*. The read and write operations are performed in a unit of page, while the erase operation is performed at the block level. After a block is erased, all pages of the block become *clean*. Each write can only operate on a clean page; when a clean page is written, it becomes a *valid* page. To improve the write performance, SSDs use the *out-of-place write* approach. That is, to update data in a valid page, the new data are first written to a different clean page, and the original page containing old data is marked as *invalid*. Thus, a block may contain a mix of clean pages, valid pages, and invalid pages.

1.2 Performance–durability tradeoff of GC algorithms

The unique I/O characteristics of SSDs pose different design requirements from those in HDDs. Since each write of SSD must be operated on a clean page, GC is required to reclaim invalid pages. GC can be triggered, for example, when the number of clean pages drops below a predefined threshold. During GC, some blocks are chosen to be erased, and all valid pages in an erased block must first be written to a different free block before the erasure. Such additional writes introduce performance overhead to normal read/write operations. To maintain high performance, one design requirement of SSDs is to minimize the *cleaning cost*, such that a GC algorithm chooses blocks containing as few valid pages as possible for reclamation.

However, SSDs allow each block to tolerate only a limited number of erasures before becoming unusable. For instance, the number is typically 100K for single-level cell (SLC) SSDs and 10K for multilevel cell (MLC) SSDs [13]. With more bits being stored in a flash cell and smaller feature size of flash cells, the maximum number of erasures tolerable by each block further decreases, for example, to several thousands or even several hundreds for the latest 3-bits MLC SSDs [23]. Thus, to maintain high durability, another design requirement of SSDs is to maximize *wear leveling* in GC, such that all blocks should have similar numbers of erasures over time so as to avoid any “hot” blocks being worn out soon.

Clearly, there is a performance–durability tradeoff in the GC design space. Specifically, a GC algorithm with a low cleaning cost may not achieve efficient wear leveling, or vice versa. Prior study (e.g., [1]) addressed the tradeoff, but that study is mainly based on simulations. From the viewpoints of SSD practitioners, it remains an open design issue of how to choose the “*best*” parameters of a GC algorithm to adapt to different tradeoff requirements for different application needs. However, understanding the performance–durability tradeoff is nontrivial, since it depends on the I/O dynamics of an SSD, and the dynamics characterization becomes complicated with the increasing numbers of blocks/pages of the SSD. This motivates us to formulate a framework that can efficiently capture the optimal design space of GC algorithms and guide the choices of parameterizing a GC algorithm to fit any tradeoff requirement.

1.3 Our contributions

In this paper, we develop a stochastic Markov chain model to characterize the I/O dynamics of an SSD, and then derive the optimal performance–durability tradeoff of a GC algorithm. Using our model as a baseline, we propose a *tunable* GC algorithm for different performance–durability tradeoff requirements. To summarize, our paper makes the following contributions:

- We formulate a stochastic Markov chain model that captures the I/O dynamics of an SSD. Since the state space of our stochastic model increases with the SSD size, we adapt the *mean-field technique* [5,41] to make the model tractable. We formally prove the convergence results under the uniform workload to enable us to analyze the steady-state performance of a GC algorithm. We also discuss how our system model can be extended for a general workload.
- We identify the optimal extremal points that correspond to the minimum cleaning cost and the maximum wear leveling, as well as the optimal tradeoff curve of cleaning cost and wear leveling that enables us to explore the *full* design space of the GC algorithms.

- Based on our analytic model, we propose a novel GC algorithm called the randomized greedy algorithm (RGA) that can be tunable to attain the operational points that follow closely along the optimal tradeoff curve. RGA also introduces low RAM usage and low computational cost.
- To address the practicality of our work, we conduct extensive simulations using the DiskSim simulator [8] with SSD extensions [1]. We first validate via synthetic workloads that our model efficiently characterizes the asymptotic steady-state performance. Furthermore, we consider real-world workload traces and use trace-driven simulations to study the performance tradeoff and versatility of RGA.

The rest of the paper proceeds as follows. In Sect. 2, we propose a Markov model to capture the system dynamics of an SSD and conduct the mean-field analysis. We formally prove the convergence, and further extend the model for a general workload. In Sect. 3, we study the design tradeoff between cleaning cost and wear leveling of GC algorithms. In Sect. 4, we propose RGA and analyze its performance. In Sect. 5, we validate our model via simulations. In Sect. 6, we present the trace-driven simulation results. In Sect. 7, we review related work, and finally in Sect. 8, we conclude the paper.

2 System model

We formulate a Markov chain model to characterize the I/O dynamics of an SSD under the read, write, and GC operations. We then analyze the model via the *mean-field technique* when the SSD scales with the increasing number of blocks or storage capacity.

2.1 Markov chain model formulation

Our model considers an SSD with N blocks of k pages each, where the typical value of k is 64/128/256 for today's commonly used SSDs [1,13,40]. Since SSDs use the out-of-place write approach (see Sect. 1), a write to a logical page may reflect on any physical page. Therefore, SSDs implement *address mapping* to map a logical page to a physical page. Address mapping is maintained in the software flash translation layer (FTL) in the SSD controller. It can be implemented in block level [46], page level [24], or hybrid form [16,34,43]. A survey of the FTL design including the address mapping mechanisms can be found in [17]. In this paper, our model abstracts out the complexity due to address mapping; specifically, we focus on the physical address space and directly characterize the I/O dynamics of physical blocks.

To help in understanding our model, we elaborate the mechanisms of handling I/O operations in an SSD. Reads and writes are file-system-level requests that are performed in units of pages. To read a page, an SSD simply shifts data out from the flash memory. However, a write operation is more complicated. Since data can only be written to clean flash pages, SSDs adopt out-of-place overwrites, that is, to update a page, an SSD first writes new data to another clean page, which is done by a flash-level *program* operation, and then it marks the original page containing

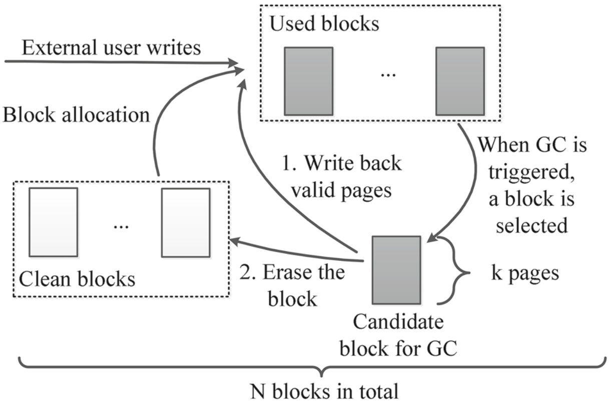


Fig. 1 The GC process in an SSD consisting of N physical blocks with k pages each. Each page is in one of the three states: clean, valid, or invalid

old data as invalid through a flash-level *invalidate* operation. An SSD performs GC operations to reclaim free space from invalid pages. Figure 1 illustrates the GC process. Specifically, when GC is triggered, say when the number of clean blocks is below some predefined threshold, the GC algorithm selects a candidate block to reclaim, such that it first writes all valid pages in the candidate block to another clean block, and then erases the candidate block and resets all pages in the block as clean. Note that the erase operation must be performed in units of blocks. In summary, the flash-level I/O requests can be classified into four types: (1) read a page, (2) perform GC on a block, (3) program (i.e., write) new data to a page, and (4) invalidate a page. We see that reads do not change the amount of valid data in each block. Each program request increases the number of valid pages in a block by one, while each invalidate request decreases it by one. GC works as if swapping two blocks, while still keeping the distribution of number of valid pages in each block unchanged.

Since a page can be in one of the three states *clean*, *valid* or *invalid*, we classify each block into a different type based on the number of valid pages containing in the block. Specifically, a block of type i contains exactly i valid pages. Since each block has k pages, a block can be of one of the $k+1$ types (i.e., from 0 to k valid pages). If a block is of type i, then we say it is in state i. We define the time duration of handling a single flash-level request as one time slot, and let $X_n(t)$ denote the state of block $n \in \{1, \ldots, N\}$ at time slot t ($t \in \mathbb{N}$). The state descriptor for the whole SSD is

$$\boldsymbol{X}^N(t) = (X_1(t), X_2(t), \ldots, X_N(t)), \quad t \in \mathbb{N}, \tag{1}$$

where $X_i(t) \in \{0, 1, \ldots, k\}$. Thus, the state space cardinality is $(k+1)^N$. To facilitate our analysis under the large system regime (as we will show later), we transform the above state descriptor to

$$\boldsymbol{n}^N(t) = (n_0(t), n_1(t), \ldots, n_k(t)), \quad t \in \mathbb{N}, \tag{2}$$

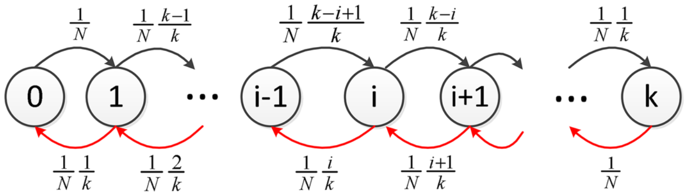


Fig. 2 State transition of a block in one time slot

where $n_i(t) \in \{0, 1, \ldots, N\}$ denotes the number of type i blocks in the SSD at time slot t. Clearly, we have $\sum_{j=0}^{k} n_j(t) = N$, and the state space cardinality is $\binom{N+k}{k}$.

We first describe how different flash-level requests affect the system dynamics of an SSD from the perspective of physical blocks. First, read requests do not change $\boldsymbol{n}^N(t)$. For GC, the SSD selects a block, writes all valid pages of that block to a clean block, and finally erases the selected block. Thus, GC requests do not change the state of $\boldsymbol{n}^N(t)$ either. On the other hand, for the program and invalidate requests, if the corresponding block is of type i, it will move from state i to state $i+1$ and to state $i-1$, respectively.

We now describe the state transition of a block in an SSD. In each time slot, we assume that only one request (either program or invalidate) arrives and triggers a state transition accordingly. Suppose that the workload is *uniform* in the sense that all pages in the SSD will have an equal probability of being accessed (in Sect. 2.5, we extend our model for a general workload). The assumption of the uniform workload implies that (1) each block has the same probability $1/N$ of being accessed, (2) the probability of invalidating one page in a block is proportional to the number of valid pages in the corresponding block, and (3) the probability of programming a page in a block is proportional to the total number of invalid and clean pages in the corresponding block. Thus, if the requested block is of type i, then the probability of invalidating one page of the block is $\frac{i}{k}$, and that of programming one page in the block is $\frac{k-i}{k}$. Figure 2 illustrates the state transitions of a single block in an SSD. If a block is at state i, each of the program and invalidate requests move the block to state $i+1$ with probability $\frac{k-i}{Nk}$ and to state $i-1$ with probability $\frac{i}{Nk}$, respectively. Note that Fig. 2 only shows the state transition of a *particular* block, but not the whole SSD. Specifically, the state space cardinality of a particular block is $k+1$ as shown in Fig. 2, while that of the whole SSD is $\binom{N+k}{k}$ as described by Eq. (2).

To characterize the I/O dynamics of an SSD, we define the *occupancy measure* $\boldsymbol{M}^N(t)$ as the vector of fraction of type i blocks at time t. Formally, we have

$$\boldsymbol{M}^N(t) = (M_0(t), M_1(t), \ldots, M_k(t)), \quad t \in \mathbb{N},$$

where $M_i(t)$ is

$$M_i(t) = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}_{\{X_n(t)=i\}} = \frac{n_i(t)}{N}, \quad t \in \mathbb{N}. \tag{3}$$

In other words, $M_i(t)$ is the *fraction* of type i blocks in the SSD. It is easy to see that the occupancy measure $\boldsymbol{M}^N(t)$ is a homogeneous Markov chain.

We are interested in modeling *large-scale* SSDs to understand the performance implication of GC algorithms. By large-scale, we mean that the number of blocks N of an SSD is large. For example, for a 256GB SSD (which is available in many of today's SSD manufactures), we have $N \approx 1 \times 10^6$ and $k = 64$ for a page size of 4 KB, implying a *huge* state space of $\boldsymbol{M}^N(t)$. Since $\boldsymbol{M}^N(t)$ does not possess any special structure (i.e., matrix-geometric form), analyzing it can be computationally expensive.

2.2 Mean-field analysis

To make our Markov chain model tractable for a large-scale SSD, we employ the *mean-field technique* [5,41]. We first introduce the concept of *intensity* denoted by $\varepsilon(N)$. Intuitively, the probability that a block performs a state transition per time slot is in the order of $\varepsilon(N)$. Under the uniform workload, since each block is accessed with the same probability $1/N$ in one time slot, we have $\varepsilon(N) = 1/N$ which vanishes as N grows. Therefore, to derive an asymptotic result, we have to re-scale the process $\boldsymbol{M}^N(t)$ to continuous time. Mathematically, the re-scaled process $\widetilde{\boldsymbol{M}}^N(\tau)(\tau \in \mathbb{R}^+)$ associated with the original process $\boldsymbol{M}^N(t)(t \in \mathbb{N})$ can be defined as follows:

$$\begin{cases} \widetilde{\boldsymbol{M}}^N(t\varepsilon(N)) = \boldsymbol{M}^N(t), \quad t \in \mathbb{N}, \\ \widetilde{\boldsymbol{M}}^N(\tau) \text{ is affine on } \tau \in [t\varepsilon(N), (t+1)\varepsilon(N)]. \end{cases} \tag{4}$$

For simplicity, in the following, we also use notations with t, e.g., $\widetilde{\boldsymbol{M}}^N(t)$, to denote continuous time processes, and we may also drop the notation t if the context is clear.

The main idea of mean-field technique is that the stochastic process $\widetilde{\boldsymbol{M}}^N(t)$ can be solved by a deterministic process $\boldsymbol{s}(t) = (s_0(t), s_1(t), \ldots, s_k(t))$ as $N \to \infty$, where $s_i(t)$ denotes the fraction of blocks of type i at time t in the deterministic process. We call $\boldsymbol{s}(t)$ the *mean-field limit*. By solving the deterministic process $\boldsymbol{s}(t)$, we can obtain the occupancy measure of the stochastic process $\boldsymbol{M}^N(t)$. We now show how the deterministic process $\boldsymbol{s}(t)$ is related to the re-scaled process $\widetilde{\boldsymbol{M}}^N(t)$. The time evolution of the deterministic process can be specified by the following set of ordinary differential equations (ODEs):

$$\begin{aligned} \frac{\mathrm{d}s_i(t)}{\mathrm{d}t} &= -s_i + \frac{k-i+1}{k}s_{i-1} + \frac{i+1}{k}s_{i+1}, \quad 1 \le i \le k-1, \\ \frac{\mathrm{d}s_0(t)}{\mathrm{d}t} &= -s_0 + \frac{1}{k}s_1, \\ \frac{\mathrm{d}s_k(t)}{\mathrm{d}t} &= -s_k + \frac{1}{k}s_{k-1}. \end{aligned} \tag{5}$$

The idea of the above ODEs is explained as follows. For an SSD with N blocks, we express the expected change in number of blocks of type i over a small time period of length dt under the re-scaled process $\widetilde{\boldsymbol{M}}^N(t)$. During this period (of length $\mathrm{d}t$ under $\widetilde{\boldsymbol{M}}^N(t)$), $N\,\mathrm{d}t$ program/invalidate requests arrive, each of which changes the state of some type i block to state $i-1$ or state $i+1$ with probability $1/N$. Since there are a total of Ns_i blocks of type i, the expected change from state i to other states is $N\,\mathrm{d}ts_i$. Using the similar arguments, the expected change in number of blocks from state $i+1$ to state i is $N\mathrm{d}t\frac{i+1}{k}s_{i+1}$, and that from state $i-1$ to state i is $N\mathrm{d}t\frac{k-i+1}{k}s_{i-1}$. Similarly, we can also specify the expected change in fraction of blocks of type 0 and type k, and we obtain the ODEs as stated in Eq. (5).

2.3 Derivation of the fixed point

We now derive the *fixed point* of the deterministic process in Eq. (5). Specifically, $\boldsymbol{s}(t)$ is said to be a fixed point if $\boldsymbol{s}(t)=\boldsymbol{\pi}$ implies $\boldsymbol{s}(t')=\boldsymbol{\pi}$ for all $t'\geq t$. In other words, the fixed point $\boldsymbol{\pi}$ describes the distribution of different types of blocks in the steady state. The necessary and sufficient condition for $\boldsymbol{\pi}$ to be a fixed point is that $\frac{\mathrm{d}\pi_i}{\mathrm{d}t}=0$ for all $i\in\{0,1,\ldots,k\}$.

Theorem 1 *Equation (5) has a unique fixed point $\boldsymbol{\pi}$ given by:*

$$\pi_i=\frac{\binom{k}{i}}{2^k},\ 0\leq i\leq k. \tag{6}$$

Proof First, it is easy to check that $\boldsymbol{\pi}$ satisfies $\frac{\mathrm{d}\pi_i}{\mathrm{d}t}=0$ for $0\leq i\leq k$. Conversely, based on the condition of $\frac{\mathrm{d}\pi_i}{\mathrm{d}t}=0$ for all i, we have

$$\begin{aligned}&-\pi_i+\frac{k-i+1}{k}\pi_{i-1}+\frac{i+1}{k}\pi_{i+1}=0,\quad 1\leq i\leq k-1,\\&-\pi_0+\frac{1}{k}\pi_1=0,\\&-\pi_k+\frac{1}{k}\pi_{k-1}=0.\end{aligned}$$

By solving these equations, we get

$$\pi_i=\binom{k}{i}\pi_k,\ \text{ for } 0\leq i\leq k.$$

Since $\sum_{i=0}^{k}\pi_i=1$, the fixed point is derived as in Eq. (6). □

Remarks Note that under uniform workload, a simpler analysis may also reach the same asymptotic occupancy measure as the states of all blocks are independent. In particular, one may analyze one block's dynamics according to the state transitions in Fig. 2, then couple N independent blocks of the same process to reach the same result in Theorem 1. However, this approach cannot be extended to analyze general workload in which the probability of accessing each block in one time slot may not be the same any more. Therefore, we perform mean-field analysis to derive the occupancy measure because of its generality.

2.4 Proof of convergence

We develop a stochastic Markov chain model to characterize the I/O dynamics of a large-scale SSD system. Specifically, we solve the stochastic process with a deterministic process via the mean-field technique and identify the fixed point in the steady state. To validate the accuracy of the derivation, we now formally prove the convergence of the SSD system state under the uniform workload. Our proof consists of two parts. We first prove that the stochastic process $\boldsymbol{M}^N(t)$ indeed converges to the deterministic process $\boldsymbol{s}(t)$ when $N \to \infty$. We then prove that the deterministic process described in Eq. (5) converges to the unique fixed point $\boldsymbol{\pi}$ in Eq. (6).

We first show that the re-scaled process $\widetilde{\boldsymbol{M}}^N(t)$ converges to $\boldsymbol{s}(t)$. Let us first show several important properties of the stochastic process $\boldsymbol{M}^N(t)$.

Lemma 1 *Define* $S = \{\boldsymbol{m} \in R^{k+1} | \sum_{i=0}^{k} m_i = 1, m_i \geq 0 \, \forall i\}$. *For any* $\boldsymbol{s} \in S$, *let* $\boldsymbol{f}^N(\boldsymbol{s}) = E(\boldsymbol{M}^N(t+1) - \boldsymbol{M}^N(t) | \boldsymbol{M}^N(t) = \boldsymbol{s})$ *be the expected change to the occupancy measure in one time slot, and let* $\varepsilon(N) = 1/N$. *We have* $\lim_{N\to\infty} \varepsilon(N) = 0$, *and* $\lim_{N\to\infty} \frac{\boldsymbol{f}^N(\boldsymbol{s})}{\varepsilon(N)}$ *exists for* $\forall \boldsymbol{s} \in S$.

Proof Since $\varepsilon(N) = \frac{1}{N}$, we have $\lim_{N\to\infty} \varepsilon(N) = 0$. We denote $\boldsymbol{f}^N(\boldsymbol{s})$ as $\boldsymbol{f}^N(\boldsymbol{s}) = (f_0^N(\boldsymbol{s}), f_1^N(\boldsymbol{s}), \ldots, f_k^N(\boldsymbol{s}))$. Consider the expected change in $M_i(t)$ $(1 \leq i \leq k-1)$ during one time slot. Since only one request arrives, the probability of changing a block of type i to other states is $\frac{1}{N} N s_i$, and the corresponding change in $M_i(t)$ is $-\frac{1}{N}$. A request may also change blocks of type $i+1$ (or type $i-1$) to state i, with probability being $\frac{i+1}{Nk} N s_{i+1}$ (or $\frac{k-i+1}{Nk} N s_{i-1}$) and the change in $M_i(t)$ being $\frac{1}{N}$ (or $\frac{1}{N}$). Thus, we have

$$f_i^N(\boldsymbol{s}) = -\frac{1}{N} s_i + \frac{i+1}{Nk} s_{i+1} + \frac{k-i+1}{Nk} s_{i-1}, \; 1 \leq i \leq k-1.$$

Similarly, we can also derive $f_0^N(\boldsymbol{s})$ and $f_k^N(\boldsymbol{s})$. Therefore, we have $\lim_{N\to\infty} \frac{\boldsymbol{f}^N(\boldsymbol{s})}{\varepsilon(N)} = \boldsymbol{f}(\boldsymbol{s})$, where

$$\begin{cases} f_i(\boldsymbol{s}) = -\, s_i + \dfrac{i+1}{k} s_{i+1} + \dfrac{k-i+1}{k} s_{i-1}, \quad 1 \leq i \leq k-1, \\ f_0(\boldsymbol{s}) = -\, s_0 + \dfrac{1}{k} s_1; \quad f_k(\boldsymbol{s}) = -s_k + \dfrac{1}{k} s_{k-1}. \end{cases}$$

□

Lemma 2 *Define* $W^N(t)$ *as an upper bound on the number of blocks that make a transition in time slot* t. *Then,* $W^N(t)$ *satisfies* $E(W^N(t)^2 | \boldsymbol{M}^N(t) = \boldsymbol{s}) \leq c N^2 \varepsilon(N)^2$ *where* c *is a constant.*

Proof During the time slot t, one request arrives, and it accesses a block with probability $\frac{1}{N}$. Therefore, $W^N(t)$ follows a binomial distribution with parameters $\frac{1}{N}$ and N.

$$E(W^N(t)^2|\boldsymbol{M}^N(t)=\boldsymbol{s})=\left(N\frac{1}{N}\right)^2+N\frac{1}{N}\left(1-\frac{1}{N}\right)\leq 2,$$

which shows the result in Lemma 2 with $c = 2$. □

Lemma 3 *There exists $\beta > 0$ and a function $\varphi(\boldsymbol{s}, \alpha)$ defined on $S \times [0, \beta]$ such that φ has continuous derivatives everywhere and $\frac{\boldsymbol{f}^N(\boldsymbol{s})}{\varepsilon(N)} = \varphi(\boldsymbol{s}, \frac{1}{N})$.*

Proof According to the proof of Lemma 1, we can derive $\frac{\boldsymbol{f}^N(\boldsymbol{s})}{\varepsilon(N)}$ as follows:

$$\begin{cases} \dfrac{f_i^N(\boldsymbol{s})}{\varepsilon(N)} = -s_i + \dfrac{i+1}{k}s_{i+1} + \dfrac{k-i+1}{k}s_{i-1}, \quad 1 \leq i \leq k-1, \\ \dfrac{f_0^N(\boldsymbol{s})}{\varepsilon(N)} = -s_0 + \dfrac{1}{k}s_1; \quad \dfrac{f_k^N(\boldsymbol{s})}{\varepsilon(N)} = -s_k + \dfrac{1}{k}s_{k-1}. \end{cases}$$

Clearly, $\frac{\boldsymbol{f}^N(\boldsymbol{s})}{\varepsilon(N)}$ is a rational function with respect to $\boldsymbol{s}$ and $\frac{1}{N}$. We let β be any number greater than $\frac{1}{N}$ and $\varphi(\boldsymbol{s}, \alpha) = \frac{\boldsymbol{f}^N(\boldsymbol{s})}{\varepsilon(N)}$, so that φ has continuous derivatives everywhere, and $\frac{\boldsymbol{f}^N(\boldsymbol{s})}{\varepsilon(N)} = \varphi(\boldsymbol{s}, \frac{1}{N})$, which completes the proof. □

Now, we can now show that the re-scaled process $\widetilde{\boldsymbol{M}}^N(t)$ converges to $\boldsymbol{s}(t)$ with the following theorem.

Theorem 2 *If $\boldsymbol{M}^N(0) \rightarrow \boldsymbol{m}$ in probability as $N \rightarrow \infty$, then for all $T > 0$, $\sup_{0\leq t\leq T} \|\widetilde{\boldsymbol{M}}^N(t) - \boldsymbol{s}(t)\| \rightarrow 0$ in probability, where $\boldsymbol{s}(t)$ satisfies the ODEs in Eq. (5) and $\boldsymbol{s}(0) = \boldsymbol{m}$.*

Proof The theorem holds due to Lemmas 1, 2, 3, and the existing theorem in [5] (Corollary 1). □

Note that the theorem in [5] provides a way to prove the convergence to mean-field limit, provided that several sufficient conditions hold. Therefore, to invoke the theorem in [5], we must explicitly verify that our model indeed satisfies the conditions, i.e., Lemma 1–3, so as to make the proof complete.

Corollary 1 *If $\boldsymbol{M}^N(0) \rightarrow \boldsymbol{m}$ in probability as $N \rightarrow \infty$, then for all $T > 0$, $\sup_{0\leq t\leq T} \|\boldsymbol{M}^N(t) - \boldsymbol{s}(\frac{t}{N})\| \rightarrow 0$ in probability, where $\boldsymbol{s}(t)$ satisfies the ODEs in Eq. (5), and $\boldsymbol{s}(0) = \boldsymbol{m}$.*

In the following, we prove that the deterministic process $\boldsymbol{s}(t)$ in Eq. (5) converges to the unique fixed point $\boldsymbol{\pi}$ in Eq. (6). The detailed proof is shown in Theorem 3. We thank Professor Benny Van Houdt for giving us invaluable comments on this proof.

Theorem 3 *The deterministic process* $\boldsymbol{s}(t)$ *which is specified by ODEs (5) converges to the fixed point* $\boldsymbol{\pi}$ *which is determined by Eq. (6).*

Proof Note that Eq. (5) can be rewritten as follows:

$$\frac{\mathrm{d}\boldsymbol{s}(t)}{\mathrm{d}t} = \boldsymbol{s}(t)\boldsymbol{Q}, \tag{7}$$

where $\boldsymbol{s}(t) = (s_0(t), s_1(t), \ldots, s_k(t))$, and $\boldsymbol{Q} = [q_{i,j}]$.

$$q_{i,j} = \begin{cases} -1, & \text{for } j = i, \\ \dfrac{k-j+1}{k}, & \text{for } i = j-1,\ j = 1, \cdots, k, \\ \dfrac{j+1}{k}, & \text{for } i = j+1,\ j = 0, \cdots, k-1. \end{cases}$$

Note that if we treat the state transition of a particular block shown in Fig. 2 as a birth–death process, then Eq. (7) exactly maps to the Kolmogorov's forward equations where $\boldsymbol{Q}$ is just the rate matrix of the birth–death process. Therefore, $\boldsymbol{s}(t)$ converges to the stationary distribution of the birth–death process π where $\boldsymbol{\pi Q} = \boldsymbol{0}$. We can easily verity that the fixed point π in Eq. (6) satisfies the condition $\boldsymbol{\pi Q} = \boldsymbol{0}$, which completes the proof. □

Our model enables us to analyze the tradeoff between cleaning cost and wear leveling of GC algorithms. As shown in Sect. 3, cleaning cost and wear leveling can be expressed as functions of $\boldsymbol{\pi}$.

2.5 Extensions to general workload

Our model thus far focused on the uniform workload, i.e., all physical pages have the same probability of being accessed. For completeness, we also generalize our model to allow for the general workload, in which blocks/pages are accessed with respect to some general probability distribution. We show how we apply the mean-field technique to approximate the I/O dynamics of an SSD, and we also conduct simulations using synthetic workloads to validate our approximation (see Sect. 5.1). As stated in Sect. 2.1, we focus on the program and invalidate requests, both of which can change the state of a block in the Markov chain model. In particular, to model the general workload, we let $p_{i,j}$ be the transition probability of a particular type i block being transited to state j due to one program/invalidate request. We have

$$p_{i,j} = 0, \quad \text{if } j \neq i-1 \text{ and } j \neq i+1,$$
$$\sum_i \sum_j p_{i,j} \left(\sum_n \mathbf{1}_{\{X_n(t)=i\}} \right) = 1,$$

where $\mathbf{1}_{\{X_n(t)=i\}}$ indicates whether block n is in state i, and thus $\sum_n \mathbf{1}_{\{X_n(t)=i\}}$ represents the number of blocks in state i. The second equation comes from the fact that each program/invalidate request can only change the state of one particular block.

In practice, $p_{i,j}$ (where $j = i - 1$ or $j = i + 1$) can be estimated via workload traces. Without loss of generality, we assume that these probabilities are not zero. Specifically, for each request being processed, one can count the number of blocks in state i (i.e., n_i) and the number of blocks in state i that change to state j (i.e., $n_{i,j}$). Then, $p_{i,j}$ can be estimated as

$$p_{i,j} \approx \frac{\sum_{\text{for each request}} \frac{n_{i,j}}{n_i}}{\text{total number of requests}}, \tag{8}$$

where $\frac{n_{i,j}}{n_i}$ is the probability that a block transits from state i to j in a particular request, and $p_{i,j}$ is the average over all requests. Note that each request only changes the state of one block, and so $p_{i,j}$ must be in the order of $O(1/N)$. Therefore, after measuring $p_{i,j}$'s, we scale them with N, which is a finite number in practical systems, and denote the corresponding re-scaled probabilities by $\tilde{p}_{i,j}$'s, or mathematically $\tilde{p}_{i,j} = N p_{i,j}$.

Applying our previous analysis framework, we can also derive the occupancy measure $\boldsymbol{M}^N(t)$ by solving a deterministic process $\boldsymbol{s}(t)$ specified by the following ODEs:

$$\begin{aligned}
\frac{\mathrm{d}s_i}{\mathrm{d}t} &= -\left(\tilde{p}_{i,i-1} + \tilde{p}_{i,i+1}\right) s_i + \tilde{p}_{i-1,i} s_{i-1} + \tilde{p}_{i+1,i} s_{i+1},\ 1 \le i \le k-1,\\
\frac{\mathrm{d}s_0}{\mathrm{d}t} &= -\tilde{p}_{0,1} s_0 + \tilde{p}_{1,0} s_1,\\
\frac{\mathrm{d}s_k}{\mathrm{d}t} &= -\tilde{p}_{k,k-1} s_k + \tilde{p}_{k-1,k} s_{k-1}.
\end{aligned} \tag{9}$$

We can further derive the fixed point of the deterministic process $\boldsymbol{s}(t)$ as in Theorem 4. Note that for the convergence proof, Theorem 3 also applies because ODEs in Eq. (9) and ODEs in Eq. (5) have the same structure.

Theorem 4 *Equation (9) has a unique fixed point $\boldsymbol{\pi}$ given by:*

$$\begin{aligned}
\pi_k &= \frac{1}{1 + \sum_{i=0}^{k-1} \frac{\prod_{j=k}^{i+1} p_{j,j-1}}{\prod_{j=i}^{k-1} p_{j,j+1}}},\\
\pi_i &= \frac{\prod_{j=k}^{i+1} p_{j,j-1}}{\prod_{j=i}^{k-1} p_{j,j+1}} \pi_k, \qquad 0 \le i \le k-1,
\end{aligned} \tag{10}$$

where $p_{i,j}$'s are measured via Eq. (8).

Proof The derivation is similar to that of Theorem 1. □

3 Design space of GC algorithms

Using our developed stochastic model, we analyze how we can parameterize a GC algorithm to adapt to different performance–durability tradeoffs. In this section, we formally define two metrics, namely *cleaning cost* and *wear leveling*, for general GC algorithms. Both metrics are defined based on the occupancy measure $\boldsymbol{\pi}$ which we derived in Sect. 2. We identify two optimal extremal points in GC algorithms. Finally, we identify the optimal tradeoff curve that explores the full optimal design space of GC algorithms.

3.1 Metrics

We now define the new parameters that are used to characterize a family of GC algorithms. When a GC algorithm is executed, it selects a block to reclaim. Let $w_i \geq 0$ (where $0 \leq i \leq k$) denote the weight of selecting a particular type i block (i.e., a block with i valid pages), such that the higher the weight w_i is, the more likely each type i block is chosen to be reclaimed. The weights are chosen with the following constraint:

$$\sum_{i=0}^{k} \frac{w_i}{N} \times n_i = \sum_{i=0}^{k} w_i \pi_i = 1. \tag{11}$$

The above constraint has the following physical meaning. The ratio w_i/N can be viewed as the probability of selecting a particular type i block for a GC operation. Since n_i is the total number of type i blocks in the system, $w_i \pi_i$ can be viewed as the probability of selecting *any* type i block for a GC operation. The summation of $w_i \pi_i$ over all i is equal to 1. Note that π_i is the occupancy measure that we derive in Sect. 2.

We now define two metrics that respectively characterize the performance and durability of a GC algorithm. The first metric is called the *cleaning cost*, denoted by $\mathcal{C}$, which is defined as the average number of valid pages contained in the block that is selected for a GC operation. This implies that the cleaning cost reflects the average number of valid pages that need to be written to another clean block during a GC operation. The cleaning cost reflects the performance of a GC algorithm, such that a high-performance GC algorithm should have a low cleaning cost. Formally, we have

$$\mathcal{C} = \sum_{i=0}^{k} i w_i \pi_i. \tag{12}$$

The second metric is called the *wear leveling*, denoted by $\mathcal{W}$, which reflects how *balanced* the blocks are being erased by a GC algorithm. To improve the durability of an SSD, each block should have approximately the same number of erasures. We simply use Jain's fairness index [29] to define the degree of wear leveling $\mathcal{W}$. Formally, we have

$$\mathcal{W} = \frac{\left(\sum_{i=0}^{k} \frac{w_i}{N} N\pi_i\right)^2}{N \sum_{i=0}^{k} \left(\frac{w_i}{N}\right)^2 N\pi_i} = \left(\sum_{i=0}^{k} w_i^2 \pi_i\right)^{-1}. \tag{13}$$

We define wear leveling based on Jain's fairness index because it effectively captures the evenness of erasures. More precisely, the higher $\mathcal{W}$ is, the more the balanced blocks that are erased. The rationale of Eq. (13) comes from the fact that $\frac{w_i}{N}$ is the probability of selecting a *particular* type i block, and there are $N\pi_i$ type i blocks in total. For example, if all w_i's are equal to one, which implies that each block has the same probability $\frac{1}{N}$ of being selected, then the wear-leveling index $\mathcal{W}$ achieves its maximum value equal to one as $\sum_{i=0}^{k} \pi_i = 1$. Note that the occupancy measure is a steady-state measure, while the occupancy of a particular physical block may change over time. Thus, the wear-leveling metric can be viewed as a measure of the evenness of physical blocks over a finite time period where the impact of physical block transitions on the wear leveling is negligible as the system size is very large. We point out that the wear-leveling metric is effective to show the design tradeoff of GC algorithms. As shown by our experiments in practical settings in Sect. 6 where the wear-leveling index is measured over the entire workload, the tradeoff relationship between the cleaning cost and wear leveling derived from our analysis still holds under real-world I/O traces.

The set of w_i's, where $0 \leq i \leq k$, will be our selection parameters to design a GC algorithm. In the following, we show how we select w_i's for different GC algorithms subject to different tradeoffs between cleaning cost and wear leveling. Our results are derived for a general workload subject to the system state distribution $\boldsymbol{\pi}$. Specifically, we also derive the closed-form solutions under the uniform workload as a case study.

3.2 GC algorithm to maximize wear leveling

Suppose that our goal is to find a set of weight w_i's such that a GC algorithm maximizes wear leveling $\mathcal{W}$. We can formulate the following optimization problem:

$$\begin{aligned} \max \quad & \mathcal{W} = \left(\sum_{i=0}^{k} w_i^2 \pi_i\right)^{-1} \\ s.t. \quad & \sum_{i=0}^{k} w_i \pi_i = 1, \\ & w_i \geq 0. \end{aligned} \tag{14}$$

The solution of the above optimization problem is to set $w_i = 1$ for all i, and the corresponding wear leveling $\mathcal{W}$ is equal to 1. Note that $\mathcal{W} \leq 1$ as it is a fairness index, and so the above solution is the optimal solution. The corresponding cleaning cost is $\sum_{i=0}^{k} i\pi_i$. In other words, each block has the same probability (i.e., $1/N$) of being selected for GC. Intuitively, this assignment strategy which maximizes wear leveling is the *random algorithm*, in which each block is uniformly chosen independent of its number of valid pages.

Under the uniform workload, we can compute the closed-form solution of the cleaning cost $\mathcal{C}$ as

$$\mathcal{C} = \sum_{i=0}^{k} i w_i \pi_i = \sum_{i=0}^{k} i \frac{\binom{k}{i}}{2^k} = \frac{k}{2}.$$

It implies that a random GC algorithm introduces an average of $k/2$ *additional page writes* under the uniform workload.

3.3 GC algorithm to minimize cleaning cost

Suppose now that our goal is to find a set of weight w_i's to minimize the cleaning cost $\mathcal{C}$, or equivalently, minimize the number of writes of valid pages during GC. The optimization formulation is

$$\begin{aligned} \min\ \ & \mathcal{C} = \sum_{i=0}^{k} i w_i \pi_i \\ s.t.\ \ & \sum_{i=0}^{k} w_i \pi_i = 1, \\ & w_i \geq 0. \end{aligned} \tag{15}$$

The solution of the above optimization problem is to set $w_0 = 1/\pi_0$ and $w_i = 0$ for all $i > 0$ (assuming that there exist some blocks of type 0), and the cleaning cost $\mathcal{C}$ is equal to 0. Since $\mathcal{C} \geq 0$, and it is equal to 0 when $w_i = 0$ for all $i > 0$, the solution is optimal. The corresponding wear leveling $\mathcal{W}$ is π_0. Intuitively, this assignment strategy corresponds to the *greedy algorithm*, which always chooses the block that has the minimum number of valid pages for GC.

Under the uniform workload, the closed-form solution of $\mathcal{W}$ corresponding to the minimum cost is given by:

$$\mathcal{W} = \frac{1}{w_0^2 \pi_0} = \frac{1}{2^k}.$$

The result shows that the greedy algorithm can significantly degrade wear leveling. For the commonly used present day's SSDs, the typical value of k is 64 or 128. This implies that the degree of wear leveling $\mathcal{W} \approx 0$, and the durability of the SSD suffers.

3.4 Exploring the full optimal design space

We identify two GC algorithms, namely the random and greedy algorithms, that correspond to two optimal extremal points of all GC algorithms. We now characterize the tradeoff between cleaning cost and wear leveling, and identify the *full* optimal design space of GC algorithms. Specifically, we formulate an optimization problem: *given a cleaning cost $\mathcal{C}^*$, what is the maximum wear leveling that a GC algorithm can achieve?* Formally, we express the problem (with respect to w_i's) as follows:

$$\begin{aligned}\max \quad & \mathcal{W} = \left(\sum_{i=0}^{k} w_i^2 \pi_i\right)^{-1} \\ s.t. \quad & \sum_{i=0}^{k} w_i \pi_i = 1, \\ & \sum_{i=0}^{k} i w_i \pi_i = \mathcal{C}^*, \\ & w_i \geq 0. \end{aligned} \tag{16}$$

Without loss of generality, we assume that $\pi_i > 0\,(0 \leq i \leq k)$. The solution of the optimization problem is stated in the following theorem:

Theorem 5 *Given a cleaning cost $\mathcal{C}^*$, the maximum wear leveling $\mathcal{W}^*$ is given by:*

$$\mathcal{W}^* = \begin{cases} \pi_0, & \mathcal{C}^* = 0, \\ \dfrac{1}{\sum_{i=0}^{\mathcal{I}} \gamma_i^2 \pi_i}, & 0 < \mathcal{C}^* < \sum_{i=0}^{k} i\pi_i, \\ 1, & \mathcal{C}^* = \sum_{i=0}^{k} i\pi_i, \\ \dfrac{1}{\sum_{i=\mathcal{L}}^{k} \Gamma_i^2 \pi_i}, & \sum_{i=0}^{k} i\pi_i < \mathcal{C}^* < k, \\ \pi_k, & \mathcal{C}^* = k, \end{cases} \tag{17}$$

for some constants γ_i, $\mathcal{I}$, Γ_i, and $\mathcal{L}$.

Proof We solve Eq. (16) by minimizing the inverse of the objective function, and the problem is a convex optimization problem. If a point $(\tilde{\boldsymbol{w}}, \tilde{\boldsymbol{u}}, \tilde{v}_1, \tilde{v}_2)$ satisfies the Karush-Kuhn-Tucker (KKT) conditions which are stated in Eq. (18), then $\tilde{\boldsymbol{w}}$ is the global minimum.

$$\begin{cases} 2w_i\pi_i - u_i + v_1\pi_i + v_2 i\pi_i = 0; \quad u_i \geq 0; \quad w_i \geq 0; \\ u_i w_i = 0; \quad \sum_{i=0}^{k} w_i\pi_i = 1; \quad \sum_{i=0}^{k} i w_i \pi_i = \mathcal{C}^*. \end{cases} \tag{18}$$

To find a point satisfying the KKT conditions, we first consider the case when $0 < \mathcal{C}^* < \sum_{i=0}^{k} i\pi_i$. Let

$$\mathcal{I}_1 = \min_{0\le j\le k}\left\{ j : \sum\nolimits_{i=0}^{j} i\pi_i - \mathcal{C}^* \sum\nolimits_{i=0}^{j} \pi_i > 0 \right\}. \tag{19}$$

Note that $\mathcal{I}_1$ must exist because $\mathcal{C}^* < \sum_{i=0}^{k} i\pi_i$ and $\sum_{i=0}^{k}\pi_i = 1$. Clearly, we have $\mathcal{I}_1 > \mathcal{C}^*$ and $\sum_{i=0}^{j} i\pi_i - \mathcal{C}^*\sum_{i=0}^{j}\pi_i > 0$ for $\mathcal{I}_1 \le j \le k$. Moreover, we have $\sum_{i=0}^{j} i^2\pi_i - \mathcal{C}^*\sum_{i=0}^{j} i\pi_i > 0\,(\mathcal{I}_1 \le j \le k)$. Now we prove that the following inequality holds.

$$\frac{\sum_{i=0}^{\mathcal{I}_1} i^2\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}_1} i\pi_i}{\sum_{i=0}^{\mathcal{I}_1} i\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}_1} \pi_i} > \mathcal{I}_1. \tag{20}$$

To prove the inequality (20), we rewrite the left hand side of the inequality as follows:

$$\frac{\sum_{i=0}^{\mathcal{I}_1} i^2\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}_1} i\pi_i}{\sum_{i=0}^{\mathcal{I}_1} i\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}_1} \pi_i} \triangleq \frac{-ax+by+\mathcal{I}_1 z}{-x+y+z},$$

where $x = -\sum_{i=0}^{\lfloor\mathcal{C}^*\rfloor}(i-\mathcal{C}^*)\pi_i$, $y = \sum_{i=\lfloor\mathcal{C}^*\rfloor+1}^{\mathcal{I}_1-1}(i-\mathcal{C}^*)\pi_i$ and $z = (\mathcal{I}_1-\mathcal{C}^*)\pi_{\mathcal{I}_1}$. Clearly, we have $x > 0, y \ge 0, z > 0$ and $\mathcal{I}_1 > b > a > 0$. Since $\mathcal{I}_1$ is the smallest integer which satisfies the condition in (19), we also have $-x+y < 0$ and $-x+y+z > 0$. Now, if $-ax+by \ge 0$, then inequality (20) holds. Otherwise,

$$\begin{aligned}\frac{-ax+by+\mathcal{I}_1 z}{-x+y+z} &= \mathcal{I}_1 + \frac{(\mathcal{I}_1-a)(x-y)+(b-a)y}{-x+y+z}\\ &> \mathcal{I}_1(\text{ as } \mathcal{I}_1 > b > a > 0, -x+y<0, \text{ and } -x+y+z>0).\end{aligned}$$

Now, we argue that there exists an $\mathcal{I}$ $(\mathcal{I}_1 \le \mathcal{I} \le k)$ such that

$$\begin{cases} \mathcal{I} < k \text{ and } \mathcal{I} < \dfrac{\sum_{i=0}^{\mathcal{I}} i^2\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}} i\pi_i}{\sum_{i=0}^{\mathcal{I}} i\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}} \pi_i} \le \mathcal{I}+1, \text{ or}\\ \mathcal{I} = k \text{ and } \mathcal{I} < \dfrac{\sum_{i=0}^{\mathcal{I}} i^2\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}} i\pi_i}{\sum_{i=0}^{\mathcal{I}} i\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}} \pi_i}. \end{cases} \tag{21}$$

To prove it, we can examine from $\mathcal{I}_1$. Since inequality (20) holds, if $\mathcal{I}_1 < k$ and $\frac{\sum_{i=0}^{\mathcal{I}_1} i^2\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}_1} i\pi_i}{\sum_{i=0}^{\mathcal{I}_1} i\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}_1} \pi_i} > \mathcal{I}_1+1$, then we have

$$\frac{\sum_{i=0}^{\mathcal{I}_1+1} i^2\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}_1+1} i\pi_i}{\sum_{i=0}^{\mathcal{I}_1+1} i\pi_i - \mathcal{C}^*\sum_{i=0}^{\mathcal{I}_1+1} \pi_i} > \mathcal{I}_1+1.$$

Therefore, either we find an $\mathcal{I}$ such that $\mathcal{I}<\frac{\sum_{i=0}^{\mathcal{I}}i^2\pi_i-\mathcal{C}^*\sum_{i=0}^{\mathcal{I}}i\pi_i}{\sum_{i=0}^{\mathcal{I}}i\pi_i-\mathcal{C}^*\sum_{i=0}^{\mathcal{I}}\pi_i}\leq\mathcal{I}+1$ or we reach k. Now, given the $\mathcal{I}$ in Eq. (21), we define

$$\begin{cases} X_{\mathcal{I}}=\sum_{i=0}^{\mathcal{I}}i^2\pi_i-\mathcal{C}^*\sum_{i=0}^{\mathcal{I}}i\pi_i,\ Y_{\mathcal{I}}=\sum_{i=0}^{\mathcal{I}}i\pi_i-\mathcal{C}^*\sum_{i=0}^{\mathcal{I}}\pi_i, \\ Z_{\mathcal{I}}=\sum_{i=0}^{\mathcal{I}}\pi_i\sum_{i=0}^{\mathcal{I}}i^2\pi_i-\left(\sum_{i=0}^{\mathcal{I}}i\pi_i\right)^2. \end{cases}$$

By Cauchy's Inequality, we have $Z_{\mathcal{I}}>0$. If we define

$$\gamma_i=X_{\mathcal{I}}/Z_{\mathcal{I}}-i\times Y_{\mathcal{I}}/Z_{\mathcal{I}}, \tag{22}$$

then we have $\gamma_i>0$, for $0\leq i\leq\mathcal{I}$, and $\gamma_i\leq 0$, for $\mathcal{I}+1\leq i\leq k$.

We can verify that $(\tilde{\boldsymbol{w}},\tilde{\boldsymbol{u}},\tilde{v}_1,\tilde{v}_2)$, which when defined as follows, satisfies the KKT conditions (18). Thus, $\tilde{\boldsymbol{w}}$ is the global minimum.

$$\begin{cases} \tilde{v}_1=-2X_{\mathcal{I}}/Z_{\mathcal{I}}, \\ \tilde{v}_2=2Y_{\mathcal{I}}/Z_{\mathcal{I}}, \end{cases} \begin{cases} \tilde{w}_i=\gamma_i,\ \ \tilde{u}_i=0,\ 0\leq i\leq\mathcal{I}, \\ \tilde{w}_i=0,\ \ \tilde{u}_i=-2\gamma_i\pi_i,\ \mathcal{I}+1\leq i\leq k, \end{cases}$$

Similarly, we can find the optimal solution for the case when $\mathcal{C}^*>\sum_{i=0}^{k}i\pi_i$. Since the framework of the proof is very similar, we only present the solution. In particular, we define

$$\begin{cases} X_{\mathcal{L}}=\sum_{i=\mathcal{L}}^{k}i^2\pi_i-\mathcal{C}^*\sum_{i=\mathcal{L}}^{k}i\pi_i,\ Y_{\mathcal{L}}=\sum_{i=\mathcal{L}}^{k}i\pi_i-\mathcal{C}^*\sum_{i=\mathcal{L}}^{k}\pi_i, \\ Z_{\mathcal{L}}=\sum_{i=\mathcal{L}}^{k}\pi_i\sum_{i=\mathcal{L}}^{k}i^2\pi_i-\left(\sum_{i=\mathcal{L}}^{k}i\pi_i\right)^2, \end{cases}$$

where $\mathcal{L}$ is an integer which satisfies the following condition:

$$\mathcal{L}>0 \text{ and } \mathcal{L}>X_{\mathcal{L}}/Y_{\mathcal{L}}\geq\mathcal{L}-1,\ \text{ or } \mathcal{L}=0 \text{ and } \mathcal{L}>X_{\mathcal{L}}/Y_{\mathcal{L}}. \tag{23}$$

If we define

$$\Gamma_i=X_{\mathcal{L}}/Z_{\mathcal{L}}-i\times Y_{\mathcal{L}}/Z_{\mathcal{L}}, \tag{24}$$

then we can also verify that $(\tilde{\boldsymbol{w}},\tilde{\boldsymbol{u}},\tilde{v}_1,\tilde{v}_2)$, which when defined as follows, satisfies the KKT conditions. Therefore, $\tilde{\boldsymbol{w}}$ is the global minimum.

$$\begin{cases} \tilde{v}_1=-2X_{\mathcal{L}}/Z_{\mathcal{L}}, \\ \tilde{v}_2=2Y_{\mathcal{L}}/Z_{\mathcal{L}}, \end{cases} \begin{cases} \tilde{w}_i=0,\ \ \tilde{u}_i=-2\Gamma_i\pi_i,\ 0\leq i\leq\mathcal{L}-1, \\ \tilde{w}_i=\Gamma_i,\ \ \tilde{u}_i=0,\ \mathcal{L}\leq i\leq k. \end{cases}$$

The cases when $\mathcal{C}^*=0$ or k and $\mathcal{C}^*=\sum_{i=0}^{k}i\pi_i$ correspond to the greedy and random algorithms, respectively. Therefore, the maximum wear leveling $\mathcal{W}^*$ can be derived as in Eq. (17) where γ_i, $\mathcal{I}$, Γ_i, and $\mathcal{L}$ are defined by Eqs. (21)–(24). □

3.5 Write amplification

To thoroughly understand the performance of GC algorithms, we have defined two metrics, cleaning cost and wear leveling. In particular, we focus on studying the tradeoff between these two performance measures so as to explore the design space of GC algorithms. Different from our analysis, some researchers study GC algorithms by deriving the *write amplification* [9,18,28,48], which is defined as the average number of physical page writes per user page write. Let A denote the write amplification cost. Here, physical page writes include the page writes initiated by external I/O, i.e., the user page writes, as well as the page writes caused by garbage collection. In other words, an SSD actually performs A internal page writes for handling every external user page write. Clearly, $A \geq 1$, and the lower the write amplification is, the higher the write performance an SSD can achieve. Note that the cleaning cost $\mathcal{C}$ represents the average number of valid pages that need to be written to another block for each GC operation; it thus implies that for every $k - \mathcal{C}$ user page writes, where k is the total number of pages in each block, $\mathcal{C}$ additional page writes are required due to GC. Therefore, we can easily derive the write amplification based on the cleaning cost. Mathematically, we have

$$A = \frac{k}{k - \mathcal{C}}. \tag{25}$$

Note that according to Eq. (25), write amplification can be regarded to be equivalent to the cleaning cost, and it only characterizes one aspect of the performance of GC algorithms. Moreover, if $\mathcal{C} = 0$, then $A = 1$, which shows the case of optimal write performance where no additional write is required for GC.

4 Randomized greedy algorithm

In this section, we present a *tunable* GC algorithm called the RGA that can operate at any given cleaning cost $\mathcal{C}^*$ and return the corresponding wear leveling close to the optimal wear leveling $\mathcal{W}^*$; or equivalently, the operational points of RGA follow closely along the optimal tradeoff curve of $\mathcal{C}^*$ and $\mathcal{W}^*$.

4.1 Algorithm details

Algorithm 1 shows the pseudo-code of RGA, which operates as follows. Each time when GC is triggered, RGA *randomly* chooses d out of N blocks $b_1, b_2, \cdots, b_d$ as candidates (Step 2). Let $v(b_i)$ denote the number of valid pages of block b_i. Then, RGA selects the block b^* that has the smallest number of valid pages, or the minimum $v(.)$, to reclaim (Step 3). We then invalidate block b^* and move its valid pages to another clean block (Steps 4–5). In essence, we define a *selection window* of window size d that defines a random subset of d out of N blocks to be selected. The window size d is the tunable parameter that enables us to choose between the random and greedy policies. Intuitively, the random selection of d blocks allows us to maximize wear leveling, while the greedy selection within the selection window allows us to minimize the cleaning cost. Note that in the special cases where $d = 1$ (resp. $d \to \infty$), RGA corresponds to the random (resp. greedy) algorithm.

Algorithm 1 Randomized greedy algorithm (RGA)

```
if garbage collection is triggered then
    randomly choose $d$ blocks $b_1, b_2, \ldots, b_d$;
    find block $b^* = \min_{v(b_i)}\{b_i : b_i \in \{b_1, b_2, \ldots, b_d\}\}$;
    write all valid pages in $b^*$ to another clean block;
    erase $b^*$;
end if
```

4.2 Performance analysis of RGA

We now derive the cleaning cost and wear leveling of RGA. We first determine the values of weights w_i's for all i. Recall from Sect. 3.1 that $w_i \pi_i$ represents the probability of choosing any block of type i for GC. In RGA, a type i block is chosen for GC if and only if the randomly chosen d blocks all contain at least i valid pages and at least one of them contains i valid pages. Thus, the corresponding probability $w_i \pi_i$ is $(\sum_{j=i}^{k} \pi_j)^d - (\sum_{j=i+1}^{k} \pi_j)^d$. Note that this expression assumes that d blocks are chosen uniformly at random from the N blocks with replacement, while in RGA, these d blocks are chosen uniformly at random without replacement. However, we can still use it as approximation since d is much smaller than N for a large-scale SSD. Therefore, we have

$$w_i = \frac{\left(\sum_{j=i}^{k} \pi_j\right)^d - \left(\sum_{j=i+1}^{k} \pi_j\right)^d}{\pi_i}. \tag{26}$$

Based on the definitions of cleaning cost $\mathcal{C}$ in Eq. (12) and wear leveling $\mathcal{W}$ in Eq. (13), we can derive $\mathcal{C}$ and $\mathcal{W}$:

$$\mathcal{C} = \sum_{i=0}^{k} i \left(\left(\sum_{j=i}^{k} \pi_j \right)^d - \left(\sum_{j=i+1}^{k} \pi_j \right)^d \right), \tag{27}$$

$$\mathcal{W} = \frac{1}{\sum_{i=0}^{k} \left(\frac{\left(\sum_{j=i}^{k} \pi_j\right)^d - \left(\sum_{j=i+1}^{k} \pi_j\right)^d}{\pi_i} \right)^2 \pi_i}. \tag{28}$$

In Sect. 5, we numerically show the relationship between cleaning cost $\mathcal{C}$ and wear leveling $\mathcal{W}$ so as to justify the efficiency of RGA. We find that the operational points of RGA follow very closely along the optimal tradeoff curve of $\mathcal{C}$ and $\mathcal{W}$, and can be easily tuned to balance the tradeoff.

4.3 Deployment of RGA

We now highlight the practical implications when RGA is deployed. RGA is implemented in the *SSD controller* as a GC algorithm. From our evaluation (see details in Sect. 5), a small value of d (which is significantly less than the number of blocks N) suffices to make RGA operate closely along the optimal tradeoff curve. This allows RGA to incur low RAM usage and low computational overhead. Specifically, RGA only needs to load the meta-information (e.g., number of valid pages) of d blocks into RAM for comparison. With a small value of d, RGA consumes an only small amount of RAM space. Also, RGA only needs to compare d blocks to select the block with the minimum number of valid pages for GC. The computational cost is $O(d)$ and hence very small as well. Since a practical SSD controller typically has limited RAM space and computational power, we expect that RGA addresses the practical needs and can be readily deployed.

We expect that RGA, like other GC algorithms, is only executed periodically or when the number of free blocks drops below a predefined threshold. The window size d can be tunable at different times during the lifespan of the SSD to achieve different levels of wear leveling and cleaning cost closely along the optimal tradeoff curve. In particular, we emphasize that the window size d can be chosen as a *non-integer*. In this case, we can simply linearly extrapolate d between $\lfloor d \rfloor$ and $\lfloor d+1 \rfloor$. Formally, for a given non-integer value d, when GC is triggered, RGA can set the window size as $\lfloor d \rfloor$ with probability p and set the window size as $\lfloor d+1 \rfloor$ with probability $1-p$, where p is given by

$$d = p\lfloor d \rfloor + (1-p)\lfloor d+1 \rfloor. \tag{29}$$

Thus, we can evaluate the values of w_i's as follows:

$$w_i(d) = pw_i(\lfloor d \rfloor) + (1-p)w_i(\lfloor d+1 \rfloor),$$

based on Eq. (26). The cleaning cost and wear leveling of RGA can be computed accordingly via Eqs. (12) and (13) substituting $w_i(d)$. More generally, we can obtain the window size from some probability distribution with the mean value given by d. This enables us to operate at *any* point close to the optimal tradeoff curve.

5 Model validation

We thus far formulate an analytic model that characterizes the I/O dynamics of an SSD, and further propose RGA that can be tuned to realize different performance–durability tradeoffs. In this section, we validate our theoretical results developed in prior sections. First, we validate via simulation that our system state derivations in Theorem 4 provide accurate approximation even for a general workload. Also, we validate the efficiency of RGA by showing that its operational points follow closely along the optimal tradeoff curve characterized in Theorem 5.

5.1 Validation on fixed-point derivations

Recall from Sect. 2 that we derived, via the mean field analysis, the fixed-point $\boldsymbol{\pi}$ for the system state of our model under both uniform and general workloads. We now validate the accuracy of such derivation. We use the DiskSim simulator [8] with SSD extensions [1]. We generate synthetic workloads for different read/write patterns to drive our simulations, and compare the system state obtained by each simulation with that of our model.

We feed the simulations with three different types of synthetic workloads: (1) Random, (2) Sequential, and (3) Hybrid. Specifically, Random means that the starting address of each I/O request is uniformly distributed in the logical address space. Note that its definition is (slightly) different from that of the uniform workload used in our model, as the latter directly considers the requests in the physical address space. The logical-to-physical address mapping will be determined by the simulator. Sequential means that each request starts at the address which immediately follows the last address accessed by the previous request. Hybrid assumes that there are 50 % of Random requests and 50 % of Sequential requests. Furthermore, for each synthetic workload, we consider both Poisson and non-Poisson arrivals. For the former, we assume that the inter-arrival time of requests follows an exponential distribution with mean 100 ms; for the latter, we assume that the inter-arrival time of requests follow a normal distribution (denoted by $N(\mu, \sigma^2)$) with mean $\mu = 100$ ms and standard deviation $\sigma = 10$ ms.

Using simulations, we generate 10 M requests for each workload and feed them to a small-scale SSD that contains 8 flash packages with 160 blocks each. We consider a small-scale SSD (i.e., with a small number of blocks) to make the SSD converge to an equilibrium state quickly with a sufficient number of requests; in Sect. 6, we consider a larger-size SSD. After running all 10 M requests, we obtain the system state of the SSD for each workload from our simulation results. On the other hand, using our model, we first execute the workload and record the transition probabilities $p_{i,j}$'s based on Eq. (8). We then compute the system state $\boldsymbol{\pi}$ using Theorem 4 for a general workload (which covers the uniform workload as well). We then compare the system states obtained from both the simulations and model derivations.

Figure 3 show the simulation and model results for the Random, Sequential, and Hybrid workloads, each associated with either the Poisson or non-Poisson arrivals of requests. The results show that under different synthetic workloads, our model derived from the mean-field technique can still provide good approximations of the system state compared with that obtained from the simulations. Note that we also observe good approximations even for non-Poisson arrivals of requests. The results show the robustness of our model in evaluating the system state.

5.2 Validation on operational points of RGA

In Sect. 3, we characterize the optimal tradeoff curve between cleaning cost and wear leveling; in Sect. 4, we present a GC algorithm called RGA that can be tuned by a parameter d to adjust the tradeoff between cleaning cost and wear leveling. We now

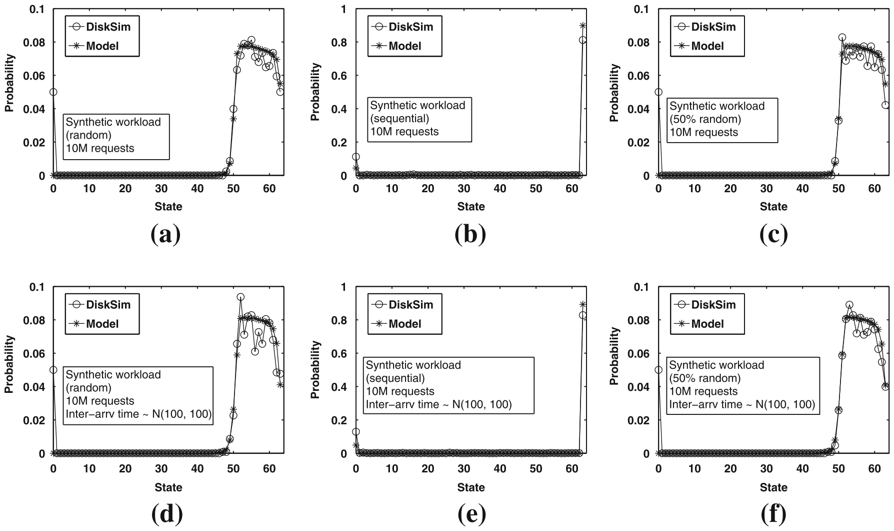


Fig. 3 Model validation on the system state $\boldsymbol{\pi}$. In each sub-figure, the x-axis represents the states (i.e., the number of valid pages in a block), and the y-axis indicates the state probabilities. **a** Random + Poisson. **b** Sequential + Poisson. **c** Hybrid + Poisson. **d** Random + non-Poisson. **e** Sequential + non-Poisson. **f** Hybrid + non-Poisson

validate that RGA can indeed be tuned to operate closely along the optimal tradeoff curve.

We consider different system state distributions $\boldsymbol{\pi}$ to study the performance of RGA. We first consider $\boldsymbol{\pi}$ derived for the uniform workload (i.e., Eq. 6). We also consider three different distributions of $\boldsymbol{\pi}$ that are drawn from truncated normal distributions, denoted by $N(\mu, \sigma^2)$ with mean μ and standard deviation σ. Figure 4a illustrates the four system state distributions, where the mean and variance of each truncated normal distribution are shown in the figure.

For each system state distribution, we compute the maximum wear leveling $\mathcal{W}^*$ for each cleaning cost $\mathcal{C}^*$ based on Theorem 5. Also, we evaluate the performance of RGA by varying the window size d from 1 to 100, and obtain the corresponding cleaning cost and wear leveling based on Eqs. (27) and (28). Here, we only focus on the integer values of d.

Figure 4b shows the results, in which the four curves represent the optimal tradeoff curves corresponding to the four different distributions of $\boldsymbol{\pi}$, while the circles correspond to the operational points of RGA with different integer values of window size d from 1 to 100. Note that the maximum wear leveling corresponds to RGA with window size $d = 1$ (i.e., the random algorithm). As the window size increases, the wear leveling decreases, while the cleaning cost also decreases. We observe that RGA indeed operates along the optimal tradeoff curves with regard to different system state distributions.

It is important to note that we can realize non-integer window sizes to further fine-tune RGA along the optimal tradeoff curve (see Sect. 4.3). To validate, we consider

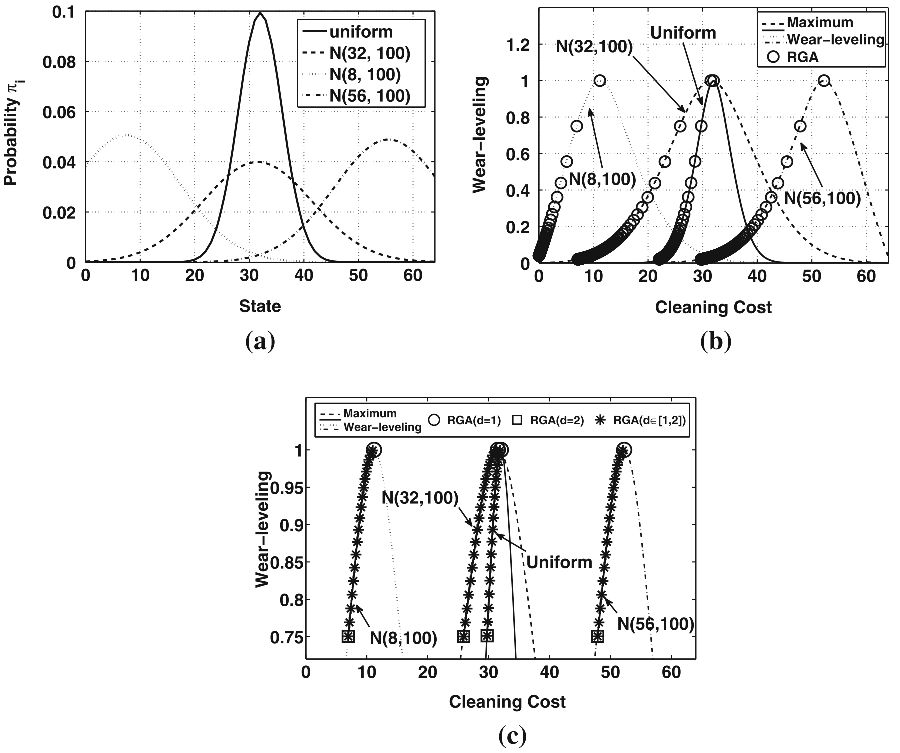


Fig. 4 Full design space and the performance of RGA. **a** Distributions of $\boldsymbol{\pi}$. **b** Design space and performance of RGA for different integers of d from 1 to 100. **c** Design space and performance of RGA for different non-integers of d from 1 to 2

different values of d from 1 to 2, with step size 0.05, and calculate d via linear extrapolation between 1 and 2.

Figure 4c shows the results for non-integer d using different system state distributions. Here, we zoom into the wear-leveling values from 0.75 to 1. Each star corresponds to the RGA with a non-integer window size obtained by Eq. (29). We observe that RGA can be further fine-tuned to operate closely along the optimal tradeoff curves even when d is a non-integer.

6 Trace-driven evaluation

In this section, we evaluate the performance of RGA under more realistic settings. Since today's SSD controllers are mainly proprietary firmware, it is nontrivial to implement GC algorithms inside a real-world SSD controller. Thus, similar to Sect. 5, we conduct our evaluation using the DiskSim simulator [8] with SSD extensions [1]. This time we focus on a large-scale SSD. We consider several real-world traces, and evaluate different metrics, including cleaning cost, I/O throughput, wear leveling, and durability, for different GC algorithms. Note that the cleaning cost and wear leveling are the metrics considered in the model, while the I/O throughput and durability are the metrics related to user experience.

Using trace-driven evaluation, our goal is to demonstrate the effectiveness of RGA in practical deployment. We compare different variants of RGA with regard to different values of window size d, as well as the random and greedy algorithms. We emphasize that we are *not* advocating a particular value of d for RGA in real-world deployment; instead, we show how different values of d can be tuned along the performance–durability tradeoff.

6.1 Datasets

We first describe the datasets that drive our evaluation. Since the read requests do not influence our analysis, we focus on four real-world traces that are all write-intensive:

- Financial [47]: It is an I/O trace collected from an online transaction process application running at a large financial institution. There are two financial traces in [47], namely Financial1.spc and Financial2.spc. Since Financial2.spc is read-dominant, we only use Financial1.spc in this paper.
- Webmail [49]: It is an I/O trace that describes the webmail workload of a university department mail server.
- Online [49]: It is an I/O trace that describes the coursework management workload on Moodle at a university.
- Webmail+Online [49]: It is the combination of the I/O traces of Webmail and Online.

Table 1 summarizes the statistics of the traces. The original Financial trace in [47] contains 24 application-specific units (ASUs) of a storage server (denoted by ASU0 to ASU23). We study the traces of all ASUs except ASU1, ASU3, and ASU5, maximum logical sector numbers of which go beyond the logical address space in our configured SSD (see Sect. 6.2). The remaining Financial trace contains around 4.4 million I/O requests, in which 77.82 % are write requests and the remaining are read requests. Also, 1.67 % of I/O requests are *sequential requests*, each of which has its starting address immediately following the last address of its prior request. The average size of each request is 5.4819 KB, meaning that most requests only access one page as the size of one page is configured as 4 KB in the simulation. The average inter-arrival time of two continuous requests is just around 10 ms. On the other hand, for the Webmail,

Table 1 Workload statistics of traces

Trace	Total no. of requests (M)	Write ratio	Sequential ratio	Avg. request size (KB)	Avg. inter-arrival time (ms)
Financial	4.4	0.7782	0.0167	5.4819	9.9886
Webmail	7.8	0.8186	0.7868	4	222.118
Online	5.7	0.7388	0.7373	4	303.763
Webmail+Online	13.5	0.7849	0.7597	4	128.302

Online and Webmail+Online traces obtained from [49], the write requests account for around 80 % of I/O requests, and over 70 % of I/O requests are sequential requests. Moreover, all requests in those traces have size 4 KB (i.e., only one page is accessed in each request), and the average inter-arrival time is much longer than that of the Financial trace. In summary, the Financial trace has the *random-write-dominant* access pattern, while the Webmail, Online, and Webmail+Online traces have the *sequential-write-dominant* access pattern.

We set the page size of an SSD as 4 KB (the default value in most today's SSDs). Since the block size considered by these traces is 512 bytes, we align the I/O requests of these traces to be multiples of the 4 KB page size. To enable us evaluate different GC algorithms, we need to make the blocks in an SSD undergo a sufficient number of program-erase cycles. However, these traces may not be long enough to trigger enough block erasures. Thus, we propose to *replay* a trace; that is, in each replay cycle, we make a copy of the original trace without changing its I/O patterns, while we only change the arrival times of the requests by adding a constant value. In our simulations, we replay the traces multiple times so that each trace file contains around 50M I/O requests. Since we replay a trace, we issue the same write request to a page multiple times, and this keeps invalidating pages due to out-of-place writes. Thus, many GC operations will be triggered, and this enables us stress-test the cleaning cost and wear-leveling metrics. We point out that this replay approach has also been used in the prior SSD study [42].

6.2 System configuration

Table 2 summarizes the parameters that we use to configure an SSD in our evaluation. We use the default configurations from the simulator whose parameters are based on a common SLC SSD [13]. Specifically, the SSD contains 8 flash packages, each of which has its own control bus and data bus, so they can process I/O requests in

Table 2 Configuration parameters

Parameter	Value
Page size	4 KB
No. of pages per block	64
No. of blocks per package	16,384
No. of packages per SSD	8
SSD capacity	32 GB
Read one page	0.025 ms
Write one page	0.2 ms
Erase one block	1.5 ms
Transfer one byte	0.000025 ms
Over-provisioning	15 %
Threshold of triggering GC	5 %

parallel. Each flash package contains 8 planes containing 2048 blocks each. Each block contains 64 pages of size 4 KB each. Therefore, each flash package contains 16384 physical blocks in total, and the physical capacity of the SSD is 32GB. For the timing parameters, the time to read one page from the flash media to the register in the plane is 25μs, and the time of programming one page from the register in the plane to the flash media is 0.2 ms. For an erase operation, it takes 1.5 ms to erase one block. The time of transferring one byte through the data bus line is 0.025μs. Since an SSD is usually over-provisioned, we set the over-provisioning factor as 15 %, which means that the advertised capacity of an SSD is only 85 % of the physical capacity. Moreover, we set the threshold of triggering GC as 5 %, meaning that GC will be triggered when the number of free blocks in the system is smaller than 5 %. Since flash packages are independent in processing I/O requests, GC is also triggered independently in each flash package. In the following, we only focus on a single flash package and compare the performance of different GC algorithms.

We consider two different initial states of an SSD before we start our simulations. The first one is the *empty* state, meaning that the SSD is entirely clean, and no data have been stored. The second one is the *full* state, meaning the SSD is fully occupied with valid data, and each logical address is always mapped to a physical page containing valid data. Thus, each write request to a (valid) page will trigger an update operation, which writes the new data to a clean page and invalidates the original page. Note that the full initial state is the default setting in the simulator. In most of our simulations (Sects. 6.3–6.5), we use the full initial state as it can be viewed as "stress-testing" the I/O performance of an SSD. When we study the durability of SSDs (Sect. 6.6), we use the empty initial state as it can be viewed as the state of a brand-new SSD.

6.3 Cleaning cost

We first evaluate the cleaning cost of different GC algorithms. In particular, we execute the traces with each of the GC algorithms and record the total number of GC operations and the total number of valid pages which are written back due to GC. We then derive the cleaning cost as the average number of valid pages that are written back in each GC operation.

Figure 5 shows the simulation results. In this figure, there are four groups of bars which correspond to the Financial, Webmail, Online, and Webmail+Online traces, respectively. In each group, there are seven bars which correspond to the greedy algorithm, random algorithm, and RGA with different window sizes d. The vertical axis represents the cleaning cost that each GC algorithm incurs. In this simulation, the simulator starts from the full initial state. We can see that the greedy algorithm incurs the smallest cleaning cost that is almost 0, while the random algorithm has the highest cleaning cost that is close to the total number of pages in each block (i.e., $k = 64$). The intuition is that if the greedy algorithm is used, then for every GC operation, the block containing the smallest number of valid pages is reclaimed, which means that it only needs to read out and write back the smallest number of pages. Therefore, the cleaning cost of the greedy algorithm should be the smallest among all algorithms. Moreover, RGA provides a variable cleaning cost between the greedy and random algorithms.

Fig. 5 Cleaning cost of different GC algorithms

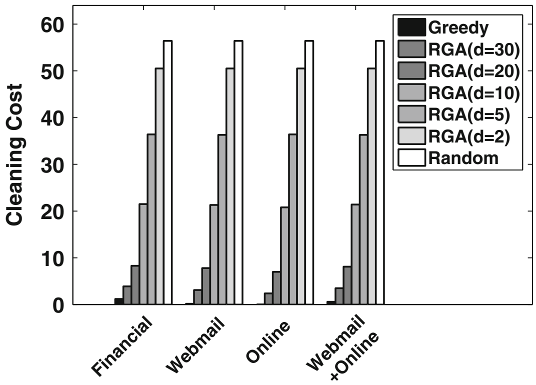
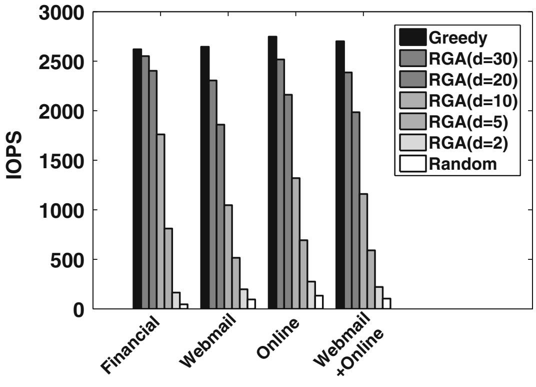


Fig. 6 IOPS of different GC algorithms



6.4 Impact on I/O throughput

We now consider the impact of different GC algorithms on the I/O throughput, using the metric Input/Output Operations Per Second (IOPS). Note that IOPS is an *indirect* indicator of the cleaning cost. Specifically, the higher the cleaning cost, the more the pages needed to be moved in each GC operation. This prolongs the duration of a GC operation, and leads to smaller IOPS as an I/O request must be queued for a longer time until a GC operation is finished.

Figure 6 shows the IOPS results of different GC algorithms (note that the simulator starts from the full initial state). We can see that the greedy algorithm achieves the highest IOPS, and the random algorithm has the lowest IOPS, which is less than 5 % of the IOPS achieved by the greedy algorithm. The results conform to those in Fig. 5. This means that the cleaning cost, the metric that we use in our analytic model, correctly reflects the resulting I/O performance. Again, RGA can provide different I/O throughput results with different values of d.

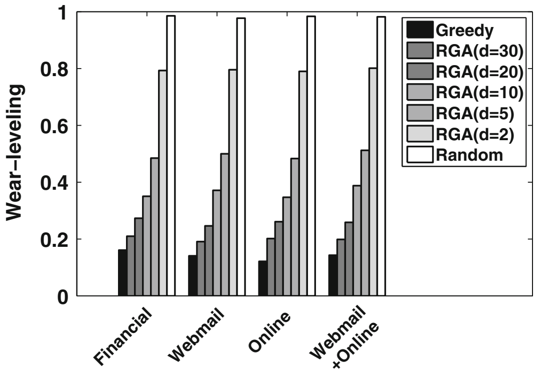


Fig. 7 Wear-leveling of different GC algorithms

6.5 Wear-leveling

We now evaluate the wear leveling of different GC algorithms. In the simulation, we execute the traces with each of the GC algorithms and record the number of times that each block has been erased. We then estimate the probability that each block is chosen for GC and derive the wear leveling based on its definition in Eq. (13).

Figure 7 shows the wear-leveling results. It is clear that the random algorithm always achieves the maximum wear leveling, which is almost one. This implies that the random algorithm can effectively balance the numbers of erasures across all blocks. On the other hand, the greedy algorithm achieves the minimum wear leveling which is less than 0.2 for all traces. Here, we note that in all traces, our RGA realizes different levels of wear leveling between the random and greedy algorithms with different values of d. In particular, when $d \leq 2$, the wear leveling of RGA is within 80 % of the maximum wear leveling of the random algorithm.

6.6 Impact on durability

The previous wear-leveling experiment provides insights into the durability (or lifetime) of an SSD. In this evaluation, we focus on examining how the durability of an SSD is affected by different GC algorithms.

To study the durability of an SSD, we have to make the SSD continue handling a sufficient number of I/O requests until it is worn out. In order to speed up our simulation, we decrease the maximum number of erasures sustainable by each block to 50. We also reduce the size of the SSD such that each flash package contains 4096 blocks, so the size of each flash package is 1GB. Other configurations are the same as we described in Sect. 6.2. Also, instead of using the real-world traces as in previous simulations, we drive the simulation with the synthetic traces that have more aggressive I/O rates so that the SSD is worn out soon. Specifically, we consider the same set of synthetic traces Random, Hybrid and Sequential as described in Sect. 5.1, but here we set the mean inter-arrival time of I/O requests to be 10 ms (as opposed to 100 ms in Sect. 5.1) based on Poisson arrivals.

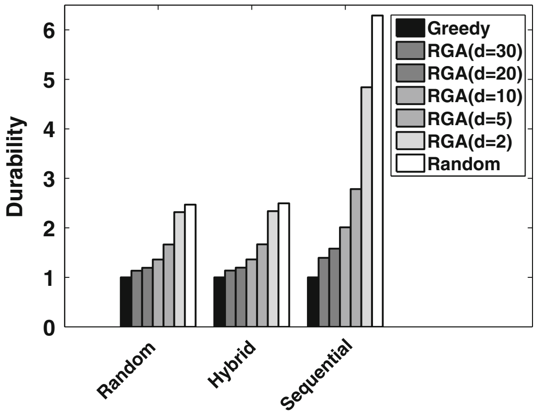


Fig. 8 Durability of different GC algorithms (normalized with respect to the greedy algorithm)

Due to the use of bad block management [39], an SSD can allow for a small percentage of bad (worn-out) blocks during its lifetime. Suppose that the SSD can allow up to $e\,\%$ of bad blocks for some parameter e. To derive the durability of the SSD, we first continue running each workload trace on the SSD until $e\,\%$ blocks are worn out, i.e., the erasure limit is reached. Then, we record the length of the duration span that the SSD survives, and take it as the durability of the SSD. For comparison, we normalize the durability with respect to that of the greedy algorithm (which is expected to have the minimum durability). In this experiment, we consider the case where $e\,\% = 5\,\%$, while we also verify that similar observations are made for other values of $e\,\% \leq 10\,\%$. Also, we assume that the SSD is brand new (i.e., the initial state is empty), and all blocks have no erasure at the beginning.

Figure 8 shows the results. We observe that the durability results of different GC algorithms are consistent with those of wear leveling in Fig. 7. We observe that the random algorithm achieves the maximum durability, and the value can be almost six times over that of the greedy algorithm (e.g., in the Sequential workload). Again, RGA provides a tunable durability between the random and greedy algorithms. When the window size $d \leq 5$, the durability of RGA can be within 68 % of the maximum lifetime of the random algorithm for Random and Hybrid workloads. For the Sequential workload, the durability of RGA drops to 40 % of the maximum lifetime of the random algorithm when $d = 5$. However, it is still almost three times higher than that of the greedy algorithm.

6.7 Summary

From the above simulations, we see that the greedy algorithm performs in the best way and the random algorithm performs the worst in terms of cleaning cost and I/O throughput, while the opposite holds in terms of wear leveling and durability. We demonstrate that our RGA provides a tradeoff spectrum between the two algorithms by tuning the window size. This simulation study not only confirms our theoretical model, but also shows that our RGA can be viewed as an effective tunable algorithm to balance between throughput performance and durability of an SSD.

7 Related studies

The research on NAND-flash based SSDs has recently received a lot of attention. Many aspects of SSDs are being studied. A survey on algorithms and data structures for flash memories can be found in [22]. Kawaguchi et al. [32] propose a flash-based file system based on the log-structured file system design. Birrell et al. [6] propose new data structures to improve the write performance of SSDs, and Gupta et al. [25] suggest to exploit value locality and design content-addressable SSDs so as to optimize the performance. Matthews et al. [38] use NAND-based disk-caching disk to mitigate the I/O bottlenecks of HDDs, and Kim et al. [33] consider hybrid storage by combining SSDs and HDDs. Agrawal et al. [1] study different design tradeoffs of SSDs via a trace-driven simulator based on DiskSim [8]. Chen et al. [13,31] further reveal many intrinsic characteristics of SSDs via empirical measurements, and Polte et al. [45] also study the performance of SSDs via experiments. Park et al. [44] mainly focus on the energy efficiency of SSDs. Li et al. [35] analyze the reliability dynamics of SSD RAID arrays. Note that [1] addresses the tradeoff between cleaning cost and wear leveling in GC, but it is mainly based on empirical evaluation.

A variety of wear-leveling techniques have been proposed, mainly from an applied perspective. Some of them are proposed in patents [2,3,7,20,26,37,50]. Several research papers have been proposed to maximize wear leveling in SSDs based on *hot–cold swapping*, main idea of which is to swap the frequently used hot data in worn blocks and the rarely used cold data in new blocks. For example, Chiang et al. [14,15] propose clustering methods for hot/cold data based on access patterns to maximize wear leveling. Jung et al. [30] propose a memory-efficient design for wear leveling by tracking only block groups, while maintaining wear-leveling performance. The authors of [10–12] also propose different strategies based on hot–cold swapping to further improve the wear-leveling performance. Our study differs from above studies in that we focus on characterizing the optimal tradeoff of GC algorithms, such that we provide flexibility for SSD practitioners to reduce wear leveling to trade for higher cleaning performance. We also propose a tunable GC algorithm to realize the tradeoff.

From a theoretical perspective, some studies propose analytic frameworks to quantify the performance of GC algorithms. A comparative study between online and offline wear-leveling policies is presented in [4]. Hu et al. [28] propose a probabilistic model to quantify the write amplification which is equivalent to the cleaning cost defined in our work. They study a modified greedy GC algorithm, and implement an event-driven simulator to validate their model. Bux and Iliadis [9] propose theoretical models to analyze the greedy GC algorithm under the uniform workload, and Desnoyers [18] also analyzes the performance of LRU and greedy GC algorithms when page-level address mapping is used. Our study differs from theirs in the following. First, the former study focuses on analyzing the write amplification which corresponds to the cleaning cost in our paper, but our focus is to analyze the tradeoff between cleaning cost and wear leveling, which are both very important in designing GC algorithms, and further explore the design space of GC algorithms. Second, our analytic models are also very different. In particular, we use a Markov model to characterize the I/O dynamics of SSDs and adapt the mean-field technique to approximate large-scale systems,and then we develop an optimization framework to derive the optimal tradeoff curve. Finally, our model also applies to general workload and address mapping, and it is further validated via trace-driven evaluation.

The study of [48] also applies the mean-field technique to analyze different GC algorithms. Its d-choices GC algorithm has the same construction as our RGA. Our study has the following key differences. First, similar to prior analytic studies, the study [48] focuses on write amplification, while we focus on the tradeoff between cleaning cost and wear leveling. Second, its analysis is limited to the uniform workload only, while we also address the general workload. Finally, we validate our analysis via trace-driven simulations, which are not considered in the study [48].

8 Conclusions

In this paper, we consider the application of Markov chain model to characterize the performance–durability tradeoff of GC algorithms in SSDs. We develop a Markov model to characterize the I/O dynamics of a large-scale SSD, and use the mean-field theory to derive the asymptotic results in equilibrium. In particular, we classify the blocks of an SSD into different types according to the number of valid pages contained in each block, and our mean-field results can provide effective approximation on the fraction of different types of blocks in steady-state even under general workload. We define two metrics, namely cleaning cost and wear leveling, to quantify the performance of GC algorithms. In particular, we theoretically characterize the optimal tradeoff curve between cleaning cost and wear leveling, and develop an optimization framework to explore the full optimal design space of GC algorithms. Taking inspiration from our analytic framework, we develop a tunable GC algorithm called the RGA which can efficiently balance the tradeoff between cleaning cost and wear leveling by tuning the parameter of the window size d. We use trace-driven simulation based on DiskSim with SSD add-ons to validate our analytic model, and show the effectiveness of RGA in tuning the performance–durability tradeoff in deployment.

Acknowledgments The work of Yongkun Li was supported in part by National Nature Science Foundation of China under Grant No. 61303048, and the Fundamental Research Funds for the Central Universities under Grant No. WK0110000040. The work of Patrick P. C. Lee was partially supported by the seed grants from the CUHK MoE-Microsoft Key Laboratory of Human-centric Computing and Interface Technologies.

References

1. Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J.D., Manasse, M., Panigrahy, R.: Design tradeoffs for SSD performance. In: Proceedings of USENIX ATC (2008) NULL
2. Assar, M., Nemazie, S., Estakhri, P.: Flash Memorymass storage architecture incorporation wear leveling technique. US patent 5,479,638 (1995)
3. Ban, A.: Wear leveling of static areas in flash memory. US patent 6,732,221 (2004)
4. Ben-Aroya, A., Toledo, S.: Competitive analysis of flash-memory algorithms. In: Proceedings of Annual European Symposium (2006)

5. Benaïm, M., Boudec, J.Y.L.: A Class of mean field interaction models for computer and communication systems. Perform. Eval. **65**(11), 823–838 (2008)
6. Birrell, A., Isard, M., Thacker, C., Wobber, T.: A design for high-performance flash disks. ACM SIGOPS Oper. Syst. Rev. **41**(2), 88–93 (2007)
7. Bruce, R.H., Bruce, R.H., Cohen, E.T., Christie, A.J.: Unified re-map and cache-index table with dual write-counters for wear-leveling of non-volitile flash Ram mass storage. US patent 6,000,006 (1999)
8. Bucy, J.S., Schindler, J., Schlosser, S.W., Ganger, G.R.: The DiskSim simulation environment version 4.0 reference manual. Tech. Rep. CMUPDL-08-101, Carnegie Mellon University (2008)
9. Bux, W., Iliadis, I.: Performance of greedy garbage collection in flash-based solid-state drives. Perform. Eval. **67**(11), 1172–1186 (2010)
10. Chang, L.P., Du, C.D.: Design and implementation of an efficient wear-leveling algorithm for solid-state-disk microcontrollers. ACM Trans. Des. Autom. Electron. Syst. **15**(1), 6:1–6:36 (2009)
11. Chang, L.P., Huang, L.C.: A Low-cost Wear-leveling Algorithm for Block-mapping Solid-state Disks. In: Proceedings of SIGPLAN/SIGBED Conference on LCTES (2011)
12. Chang, Y.H., Hsieh, J.W., Kuo, T.W.: Improving flash wear-leveling by proactively moving static data. IEEE Tran. Comput. **59**, 53–65 (2010)
13. Chen, F., Koufaty, D.A., Zhang, X.: Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives. In: Proceedings of ACM SIGMETRICS (2009)
14. Chiang, M.L., Chang, R.C.: Cleaning policies in mobile computers using flash memory. J. Syst. Softw. **48**(3), 213–231 (1999)
15. Chiang, M.L., Lee, P.C.H., Chang, R.C.: Using data clustering to improve cleaning performance for flash memory. Softw. Pract. Exp. **29**(3), 267–290 (1999)
16. Chung, T.S., Park, D.J., Park, S., Lee, D.H., Lee, S.W., Song, H.J.: System software for flash memory: a survey. In: Proceedings of International Conferences on Embedded and Ubiquitous, Computing (2006)
17. Chung, T.S., Park, D.J., Park, S., Lee, D.H., Lee, S.W., Song, H.J.: A survey of flash translation layer. J. Syst. Arch. **55**(5–6), 332–343 (2009)
18. Desnoyers, P.: Analytic modeling of SSD write performance. In: Proceedings of SYSTOR (2012)
19. Enderle, R.: Revolution in January: EMC brings flash drives into the data center. http://www.itbusinessedge.com/blogs/rob/?p=184 (2008). Accessed 29 Mar 2014
20. Estakhri, P., Assar, M., Reid, R., Alan, Iman, B.: Method of and architecture for controlling system data with automatic wear leveling in a semiconductor non-volitile mass storage memory. US patent 5,835,935 (1998)
21. Floyer, D.: Flash Pricing Trends Disrupt Storage. http://wikibon.org/wiki/v/Flash_Pricing_Trends_Disrupt_Storage (2010). Accessed 29 Mar 2014
22. Gal, E., Toledo, S.: Algorithms and data structures for flash memories. ACM Comput. Surv. **37**(2), 138–163 (2005)
23. Grupp, L.M., Davis, J.D., Swanson, S.: The bleak future of NAND flash memory. In: Proceedings of USENIX FAST (2012)
24. Gupta, A., Kim, Y., Urgaonkar, B.: DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In: Proceedings of ACM ASPLOS (2009)
25. Gupta, A., Pisolkar, R., Urgaonkar, B., Sivasubramaniam, A.: Leveraging value locality in optimizing NAND flash-based SSDs. In: Proceedings of USENIX FAST (2011)
26. Han, S.W.: Flash memory wear leveling system and method. US patent 6,016,275 (2000)
27. Hess, K.: 2011: Year of the SSD? http://www.datacenterknowledge.com/archives/2011/02/17/2011-year-of-the-ssd/ (2011). Accessed 29 Mar 2014
28. Hu, X.Y., Eleftheriou, E., Haas, R., Iliadis, I., Pletka, R.: Write amplification analysis in flash-based solid state drives. In: Proceedings of SYSTOR (2009)
29. Jain, R., Chiu, D.M., Hawe, W.: A Quantitative measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report DEC (1984)
30. Jung, D., Chae, Y.H., Jo, H., Kim, J.S., Lee, J.: A group-based wear-leveling algorithm for large-capacity flash memory storage systems. In: Proceedings of International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (2007)
31. Jung, M., Kandemir, M.: Revisiting widely held SSD expectations and rethinking system-level implications. In: Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '13, pp. 203–216. ACM (2013)
32. Kawaguchi, A., Nishioka, S., Motoda, H.: A flash-memory based file system. In: Proceedings of USENIX Technical Conference (1995)

33. Kim, Y., Gupta, A., Urgaonkar, B., Berman, P., Sivasubramaniam, A.: HybridStore: a cost-efficient, high-performance storage system combining SSDs and HDDs. In: Proceedings of IEEE MASCOTS (2011)
34. Lee, S.W., Park, D.J., Chung, T.S., Lee, D.H., Park, S., Song, H.J.: A log buffer-based flash translation layer using fully-associative sector translation. ACM Trans. Embed. Comput. Syst. **6**(3), 18 (2007)
35. Li, Y., Lee, P.P.C., Lui, J.C.S.: Stochastic analysis on RAID reliability for solid-state drives. In: Proceedings of the 32nd IEEE International Symposium on Reliable Distributed Systems (2013)
36. Li, Y., Lee, P.P.C., Lui, J.C.S.: Stochastic modeling of large-scale Solid-State Storage Systems: Analysis, Design Tradeoffs and Optimization. In: Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems, SIGMETRICS '13, pp. 179–190. ACM (2013)
37. Lofgren, K.M.J., Norman, R.D., Thelin, G.B., Gupta, A.: Wear leveling techniques for flash EEPROM systems. US patent 6,850,443 (2005)
38. Matthews, J., Trika, S., Hensgen, D., Coulson, R., Grimsrud, K.: Intel^R turbo memory: nonvolatile disk caches in the storage hierarchy of mainstream computer systems. ACM Trans. Storage **4**(2), 1–24 (2008)
39. Micron Technology: Bad block management in NAND flash memory. Technical Note, TN-29-59 (2011)
40. Micron Technology. http://www.micron.com/products/nand-flash. Accessed 29 Mar 2014
41. Mitzenmacher, M.: Load balancing and density dependent jump Markov processes. In: Proceedings of IEEE FOCS (1996)
42. Murugan, M., Du, D.: Rejuvenator: a static wear leveling algorithm for NAND flash memory with minimized overhead. In: Proceedings of IEEE MSST (2011)
43. Park, C., Cheon, W., Kang, J., Roh, K., Cho, W., Kim, J.S.: A reconfigurable ftl (flash translation layer) architecture for nand flash-based applications. ACM Trans. Embed. Comput. Syst. **7**(4), 38:1–38:23 (2008)
44. Park, S., Kim, Y., Urgaonkar, B., Lee, J., Seo, E.: A comprehensive study of energy efficiency and performance of flash-based SSD. J. Syst. Arch. **57**(4), 354–365 (2011)
45. Polte, M., Simsa, J., Gibson, G.: Enabling enterprise solid state disks performance. In: 1st Workshop on Integrating Solid-state Memory into the Storage Hierarchy (2009)
46. Qin, Z., Wang, Y., Liu, D., Shao, Z.: Demand-based block-level address mapping in large-scale NAND flash storage systems. In: Proceedings of IEEE/ACM/IFIP CODES+ISSS (2010)
47. Storage Performance Council: http://traces.cs.umass.edu/index.php/Storage/Storage (2002). Accessed 29 Mar 2014
48. Van Houdt, B.: A mean field model for a class of garbage collection algorithms in flash-based solid state drives. In: Proceedings of ACM SIGMETRICS (2013)
49. Verma, A., Koller, R., Useche, L., Rangaswami, R.: SRCMap: Energy proportional storage using dynamic consolidation. In: Proceedings of USENIX FAST (2010). http://sylab.cs.fiu.edu/projects/srcmap/start. Accessed 29 Mar 2014
50. Wells, S.E.: Method for wear leveling in a flash EEPROM memory. US patent 5,341,339 (1994)