TEL-AVIV UNIVERSITY
RAYMOND AND BEVERLY SACKLER FACULTY OF EXACT SCIENCES
SCHOOL OF COMPUTER SCIENCE

Competitive Analysis of Flash-Memory Algorithms

Thesis submitted in partial fulfillment of the requirements for the M.Sc. degree of Tel-Aviv University by

Avraham Ben-Aroya

The research work for this thesis has been carried out at Tel-Aviv University under the direction of Prof. Sivan Toledo

April 2006

Abstract

Flash memories are widely used in computer systems ranging from embedded systems to workstations and servers to digital cameras and mobile phones. The memory cells of flash devices can sonly endure a limited number of write cycles, usually between 10,000 and 1,000,000. Furthermore, cells containing data must be *erased* before they can store new data, and erasure operations erase large blocks of memory, not individual cells. To maximize the endurance of the device (the amount of useful data that can be written to it before one of its cells wears out), flash-based systems move data around in an attempt to reduce the total number of erasures and to level the wear of the different erase blocks. This data movement introduces an interesting online problem called the *wear-leveling problem*. Wear-leveling algorithms have been used at least since 1993, but they have never been mathematically analyzed. In this thesis we analyze the two main wear-leveling problems. We show that a simple randomized algorithm for one of them is essentially optimal both in the competitive sense and in the absolute sense (our competitive result relies on an analysis of a nearly-optimal offline algorithm). We show that deterministic algorithms cannot achieve comparable endurance. We also analyze a more difficult problem and show that offline algorithms for it can improve upon naive approaches, but that online algorithms essentially cannot.

Contents

Acknowledgments

I would like to thank my advisor, Prof. Sivan Toledo, for the helpful guidance he had given me through the research and for all the patience he had for me. I would also like to thank my dear parents, Eliyahou and Orna, and my sister, Tali, for their boundless love, care and support. This thesis is dedicated to them.

CHAPTER 1

Introduction

Flash memory is a type of electrically erasable programmable read-only memory (EEPROM). Flash memory is nonvolatile (retains its content without power), so it is used to store files and other persistent objects in handheld computers and mobile phones, in digital cameras, in portable music players, in workstations and servers (usually only for the boot program and its parameters), and in numerous other devices.

The read/write/erase behaviors of flash memory is radically different than that of other programmable memories, such as magnetic disks and volatile RAM. Most importantly for this thesis, memory cells in a flash device (as well as in other types of EEPROM memories) can be erased only a limited number of times, between 10,000 and 1,000,000, after which they wear out and become unreliable.

Clever management of a flash device can dramatically extend its functional life span. Consider a device with n memory cells that can each be erased H times. Most systems cannot use a device with some unreliable cells, so from the system's point of view, the device becomes useless once one of the n cells reaches the wear limit H. When the device becomes useless, it has been written to between $H + 1$ and $n(H + 1)$ times. Clever management aims to ensure that the device can be successfully written to as close to $n(H + 1)$ times as possible. Techniques that aim to achieve this goal are called in the flash literature *wear-leveling* techniques.

Wear-leveling techniques work by separating the system's naming of memory cells from the addressing of physical cells. The system views the flash device as an array of $m \leq n$ memory cells called *blocks*. The flash memory manager maps each block to a physical memory cell, called a *sector*. Occasional changes in the mapping can ensure that the wear of all sectors is roughly the same, even if the system writes to certain blocks much more than to others. If the mapping of a block is changed only when the system writes to it, then the mapping changes induce no extra writes. If, on the other hand, the flash-memory manager moves blocks from sector to sector to level the wear even when their contents is not changed, then wear-leveling induces extra writes.

Once a physical sector is written to, its contents cannot be arbitrarily changed before it is *erased*, which brings all its bits to the '1' state. Some

flash devices allow clearing '1' bits in a sector without erasing it, but in all flash devices the only way to set a '0' bit is to erase an entire group of contiguous memory cells called an *erase unit*. In traditional EEPROM, erase units are tiny, a singly byte. In modern flash memories, erase units are larger, ranging from 512 bytes to 64 KB. Large erase units allow fast bulk erasures.

When erase units are relatively small, each unit usually stores one block. In other words, the size of erase units, sectors, and blocks are all the same (except perhaps for a sector and/or erase-unit headers). Most recent high-capacity flash devices, which use a technology called NAND flash, are built this way, with erase units designed to store a 512 byte block plus a header. Smaller devices, older devices, and devices designed to be memory-mapped use a technology called NOR flash, which favors large erase units. In such devices, the system breaks each erase unit into sectors. All the sectors of a unit are erased together, but they can be written to separately. NOR devices that are used as magnetic-disk emulators usually use fixed-size sectors, but in other uses sectors often have variable size, to save space.

Starting around 1993, a variety of wear-leveling techniques have been proposed, mostly in patents [**1, 2, 13, 10, 6, 12, 16, 17, 5, 21, 4, 3, 22, 9, 14, 23, 7, 8, 15**]; for details about these techniques and about other flash-management techniques, see [**11**]. In this thesis, we present a competitive analysis of online wear-leveling policies. No such analysis has ever been published. Some of our analyses, such as the lower bounds for deterministic policies, apply directly to algorithms that have been previously proposed. The rest of the thesis analyses algorithms that are similar to proposed algorithms, but not always identical.

The thesis is organized as follows. The next chapter contains some preliminaries. Chapter explains how flash memories are managed by software systems (sometimes embedded into firmware). Chapter 4 defines a mathematical model for flash-memory management problems. Chapters 5, 6, and 7 analyze a simple but common wear-leveling problem, in which the system performs wear-leveling at the granularity of entire erase units. The three chapters discuss offline algorithms for this problem, deterministic online algorithms, and randomized online algorithms. Chapter 8 presents the results of simulations that we performed in order to quantitatively understand our randomized online algorithm and in order to provide engineers the ability to tune a key parameter of this algorithm. Chapter 9 presents a more challenging wear-leveling problem, in which the system can write to flash data blocks smaller than an erase unit. We analyze this problem in Chapters 10, 11, and 12; the three chapters discuss offline algorithms, deterministic online algorithms, and randomized online algorithms. We present our conclusions in Chapter 13.

CHAPTER 2

Preliminaries

The binomial distribution is the probability distribution of the number of success in a sequence of n independent trials, each with success probability p. This distribution is denoted by $B(n,p)$. If $X \sim B(n,p)$ then for any $0 \leq k \leq n$ it holds that

$$\Pr[X = k] = \binom{n}{k} p^k (1-p)^{n-k} .$$

The binomial distribution is highly concentrated around its mean. The deviation from the mean can be bounded using Chernoff bound. Specifically, we will need the following version of Chernoff bound, taken from [**18**]:

THEOREM. *If $X \sim B(n,p)$ then for any $\epsilon > 0$ and for any $0 < \delta \leq 1$ it holds that*

$$\Pr[X \leq (1-\epsilon)np] \leq e^{-\frac{1}{2}\epsilon^2 np} ,$$

and

$$\Pr[X \geq (1+\delta)np] \leq e^{-\frac{1}{3}\epsilon^2 np} .$$

The geometric distribution is the probability distribution of the number trials needed to get one success in a series of independent trials, each with success probability p. This distribution is denoted by $G(p)$. If $X \sim G(p)$ then for any $k \geq 1$ it holds that

$$\Pr[X = k] = (1-p)^{k-1} p .$$

For functions $f(n), g(n)$, the expression $f \ll g$ means $f = o(g)$, and $f \gg g$ means $f = \omega(g)$.

The term polylog(n) denotes the class of functions $\bigcup_{k \geq 1} O(\log^k n)$.

CHAPTER 3

Principles of Flash Management

We model the wear-leveling problem as follows. Consider a flash device with n erase units, each of which can store k sectors, which is used to store $m \leq nk$ blocks. We allow $m < nk$ for two reasons. First, spare sectors allow systems to implement atomic updates. Second, spare sectors can help even the wear in certain classes of policies. A data structure that is stored on the flash device itself maps each block to a sector. The data structure is stored on flash so that the mapping is not lost when the system is shut down. This data structure is irrelevant in this thesis.

Initially, each data block is stored in some sector. When the system needs to overwrite a block of data, it issues an update request to the flash-management software; we refer to this software as the *driver*. The request specifies the name of the block and its new contents. The driver serves the request by performing a sequence of erasure and writing operations. The sequence always needs to achieve one goal, and in most systems, it needs to achieve two more:

- At the end of the sequence each block must be stored in some sector. This is always necessary.
- The rearrangement of blocks might also contribute to wear leveling.
- Most systems require that the rearrangement of blocks is carried out such that no data is lost if the system is shut off in the middle of the sequence. This is an atomicity requirement with respect to the block-update request.

If atomicity is not an issue, the sequence always has the same structure. The driver begins the sequence by marking the sector that contains the old copy of the block as obsolete. If the driver wishes to rearrange additional blocks, it reads them into volatile memory (RAM) and marks the sectors that contained them as obsolete. Next, the driver can erase units that contain no valid sectors, only obsolete and erased ones. Finally, the driver writes all the blocks that are in volatile memory, including the updated block that initiated the sequence, into erased sectors. If atomicity is required, update sequences are typically more complex. Erasures might be interleaved with read and write operations, to ensure that an erased unit contains no blocks that are not stored elsewhere.

The driver might not have enough RAM to store in memory many blocks. Many flash-management policies, both policies that have been proposed in the literature or in patents [**11**] and policies that we analyze in this thesis, do not require much memory. For such policies, RAM usage is not an important issue. When we discuss policies that do require a large RAM, we note this, but we do not formulate a specialized RAM-limited competitive model.

Another issue that we ignore in this thesis are variable-length blocks. Some flash drivers support them, but very few (if any) can arbitrarily move variable-length blocks between erase units. Allowing such flexibility complicates the data structure that keeps track of the location of blocks. In most systems that use variable-length blocks, the blocks on a single erase units are managed by the driver as a single large block ($k = 1$).

CHAPTER 4

A Mathematical Model of Wear Leveling

We model the flash device as an array of n bins, labeled 1 through n. Each bin contains k slots. We denote the jth slot in the ith bin by $i.j$. If $k = 1$, we do not distinguish between slots and bins and use a single index i to denote a bin/slot. We refer to the case $k = 1$ as the *unit bins* case. Bins model erase units, and slots model sectors. We model the blocks of data that the system stores in the flash device as m named balls, with names 1 through m. Slots are in one of three possible states. If a ball is stored in a slot, the slot is *occupied*. Otherwise, a slot is either *clean* or *dirty*. A clean slot does not currently store any ball, but it can store any ball. A dirty slot cannot store a ball. To clean a slot, the bin that contains the slot must be *erased*. An erasure cleans all the slots in the bin. In particular, if some of the slots are occupied, balls are lost. This must never happen. An occupied slot becomes dirty if the driver moves the occupying ball to another location. In an *atomic policy*, the driver is only allowed to move a ball to a clean slot. In a non-atomic policy, the driver can also move balls to and from a separate staging area that models RAM.

Initially, each ball is stored in some slot. The system issues a series of requests, each of which names a ball (a block to update). The driver must then perform a sequence of moves and erasures that satisfy the following conditions:

- At the end of the sequence, each one of the m balls occupy some slot.
- The named ball must be moved at least once (although it can end up in its original slot).

If the driver cannot serve a request without exceeding the maximal number H of erasures per unit, we consider the device worn out and destroyed. We assume that H is known (in practice, manufacturers specify a lower bound on H). When the request series begins, we have m occupied slots and $nk - m$ clean slots. Thus, the driver cannot serve more than $(nk - m) + Hnk$ requests. We can, therefore, assume that the length of all request sequences is $(nk - m) + Hnk$. The objective of the driver is to serve as many requests as possible before the device wears out.

We use competitive analysis to quantify the effectiveness of online wear-leveling policies. Let $\ell_{\mathsf{opt}}(\sigma)$ be the number of requests that the optimal off-line algorithm can serve for a given request sequence σ, and let $\ell_\alpha(\sigma)$ be the number of requests that an online algorithm α can serve. The competitive ratio of α is

$$r_\alpha = \min_\sigma \frac{\ell_\alpha(\sigma)}{\ell_{\mathsf{opt}}(\sigma)} \ , \tag{4.0.1}$$

where the minimization is over all the sequences of length $(nk - m) + Hnk$. A good online policy achieves a high competitive ratio. If α is a randomized then we replace $\ell_\alpha(\sigma)$ in (4.0.1) by the expected length that α can serve.

CHAPTER 5

Bounding The Optimal Offline Algorithm: Unit Bins

We begin the analysis with unit bins ($k = 1$). In this case we simplify the model described in the previous chapter. First, since each bin contains exactly one slot, we stop using the term slots and refer only to bins. Second, we allow a bin to be only in two states - empty and occupied. Whenever a ball is taken out of a bin, the bin is immediately erased (and thus becomes clean).

It is not hard to see that this model is closely related to the previous one: For any algorithm α in the simplified model we can create a matching algorithm α' in the previous model that preforms greedy erasures (erases a bin whenever it is dirty and a ball should be written to it). Clearly, for every request sequence σ, the number of requests α serves before a bin is erased H times (in the simplified model) is at most the number of requests α' serves before a bin is erased H times (in the previous model). Moreover, the number of requests α' serves before a bin is erased H times is at most the number of requests α serves before a bin is erased $H + 1$ times (in the simplified model). Thus, any result for the simplified model applies to the previous model up to a change of ± 1 to H. The entire analysis of the unit bins case is done in the simplified model.

In this chapter we describe and analyze an offline algorithm off that achieves $\ell_{\mathsf{off}}(\sigma) \geq Hn - n$. This allows us to estimate ℓ_{opt} to within an additive error of at most n,

$$Hn \geq \ell_{\mathsf{opt}} \geq \ell_{\mathsf{off}}(\sigma) \geq Hn - n \ .$$

We present most of the analysis in terms of a non-atomic algorithm and then show an atomic variant.

5.1. A Non-Atomic Offline Algorithm

Normally, the algorithm serves a request to ball x stored in bin i by erasing i and putting x back into i. In some cases, however, the algorithm decides to exchange the contents of i with the contents of another bin j. To decide whether to switch i with j, the algorithm counts the number of remaining requests to the ball y stored in bin j, but only up to request number $Hn-n$. It switches if the number of remaining requests to the ball y stored in j matches exactly the number of erasures left for bin i. Such

a switch always occurs during a request to either x or y; a request to x reduces the number of erasures left for i, and a request to y reduced the number of remaining requests to y. The algorithm performs at most n such switches on a given sequence. Notice that a switch requires two erasures, while a write in-place requires only one.

We now describe the algorithm more fully and more formally. We denote by $h_i(t)$ the number of erasures that the algorithm already performed on bin i immediately before serving request number t. We call $h_i(t)$ the *wear* of the bin i at time t. We denote by $r_x(t)$ the number of requests to ball x in requests t through $Hn - n$. We denote by $\text{ball}(i,t)$ the index of the ball in bin i just before request t arrives, and we denote by $\text{bin}(x,t)$ the index of the bin that ball x resides in just before request t arrives. We denote by σ_t the index of the ball requested by the tth request in the sequence σ.

If $r_x(t) + h_{\text{bin}(x,t)}(t) < H - 1$, we say that x is *safe* at time t. If $r_x(t) + h_{\text{bin}(x,t)}(t) > H$, we say that x is *dangerous* at time t. If $r_x(t) + h_{\text{bin}(x,t)}(t) = H$ or $r_x(t) + h_{\text{bin}(x,t)}(t) = H - 1$, we say that x is *exact* at time t. If a safe ball is left in its bin, that bin will not be erased more than $H - 2$ times until $Hn - n$ requests are served; if a dangerous ball is left in its bin, the bin will wear out; if an exact ball is left in its bin, the bin will be erased exactly or almost exactly to the wear limit.

If $m < n$, some bins are empty at all times and this requires special treatment. To unify the case of empty and full bins, we pretend that $m = n$ and that empty bins contain dummy balls that are never requested. The bound that we get is slightly weaker than the one we could get with a special treatment for the empty bins, because dummy balls do not cause erasures. However, the gap is small, $n - m$, and the analysis is simplified considerably.

The algorithm maintains a partitioning of the balls into two subsets, the set A of active balls and the set E of balls we are done with. Initially, all the exact balls are in E (i.e. each ball x, such that $r_x(1) \in \{H, H-1\}$), and the rest of the balls are in A. Suppose that we already served $t - 1$ requests. The algorithm serves $\sigma_t = \text{ball}(i,t)$ as follows.

(1) If $\sigma_t \in E$, put σ_t back into i.
(2) Otherwise, if σ_t is safe, $r_{\sigma_t}(t) + h_i(t) < H - 1$, put σ_t back into i.
(3) Otherwise, if there is a bin $j = \text{bin}(y,t)$ such that
    (a) $y \in A$
    (b) $r_y(t) + h_j(t) < H - 1$
    (c) $r_{\sigma_t}(t) + h_j(t) = H$

    then perform the following actions.
    (a) Copy y into i, put σ_t in j, and move σ_t to E.
    (b) If $r_y(t) + h_i(t) \in \{H, H-1\}$ then also move y to E.
(4) Otherwise, if there is a bin $j = \text{bin}(y,t)$ such that

(a) $y \in A$
(b) $r_y(t) + h_j(t) < H - 1$
(c) $r_y(t) + (h_i(t) + 1) = H$
then copy y into i, put σ_t in j, and move y to E.
(5) Otherwise, put σ_t back into i.

We now prove that the algorithm is correct, in the sense that after $Hn - n$ requests are served, no bin is erased more than H times. The proof is somewhat complex, so before explaining how the proof works we make a few observations. Some are trivial, and the rest will be proved later.

The set E contains all the exact balls and no other balls. Requests to exact and safe balls never trigger a switch, only requests to dangerous balls. A switch always makes at least one ball exact in the strong sense (the bin will wear completely after request $Hn - n$). Exact balls stay at their bin, so they never become inexact. Bins can only wear out under Rule 5.

The proof works as follows. The algorithm matches dangerous and safe balls to make at least one of them strongly exact. The key to the proof is showing that a dangerous ball is always matched before it wears out its bin. We show this by showing that switching *any* dangerous/safe pair of balls never causes the dangerous ball to become safe or the safe one to become dangerous. This, together with another invariant that shows that there are never only exact and dangerous balls, implies the correctness of the algorithm.

The first lemma shows that E is the set of exact balls. Like all the other results in this chapter, its conclusion holds for time $1 \leq t \leq Hn - n + 1$, that is, for the useful duration of the algorithm (the useful duration ends at $t = Hn - n + 1$, which is *after* request $Hn - n$ is served).

LEMMA 5.1.1. *At any time* $1 \leq t \leq Hn - n + 1$*, the set* E *contains exactly the set of exact balls.*

PROOF. We show that an exact x is in E by induction on t. We denote by E_t the set E at time t. For $t = 1$ the lemma follows from the fact that E_1 is defined as the set of exact balls at time 1. Let us assume correctness for time $t - 1$, and prove it for time t. Let x be exact at time t. If x was exact at time $t - 1$ then by the induction assumption we have $x \in E_{t-1}$. Since a ball can never leave the set E, we have $x \in E_t$. If x was not exact at time $t - 1$, then it became exact when σ_{t-1} was served. The only rules which can change the safety property of a ball are Rule 3 and Rule 4. In both rules x can be either σ_{t-1} or the switched ball y. We split the analysis into the four possible cases:

- If $x = \sigma_{t-1}$ and σ_{t-1} was served by Rule 3, or $x = y$ and σ_{t-1} was served by Rule 4, then x moves to E.
- If $x = y$ and σ_{t-1} was served by Rule 3, then because x is exact at time t, the condition (3b) holds, and x moves to E.

- If $x = \sigma_{t-1}$ and σ_{t-1} was served by Rule 4, then x cannot be exact at time t; if it is, then x should have been served by Rule 3 instead.

Therefore, we have $x \in E_t$ as desired.

We now need to show the other direction, that an $x \in D_t$ is exact at time t. Clearly, when a ball is moved to D it is exact, thanks to the rules of the algorithm. Since a ball in D is never switched and since an exact ball that stays put remains exact, the conclusion holds. □

The next lemma states that every switch causes at least one ball to become strongly exact. Switching performs two erasures per request, so they are relatively inefficient. This lemma will help us bound the number of switches.

LEMMA 5.1.2. *Let w_t be the number of switches performed by time t, and let f_t be the number of strongly exact balls (balls x for which $r_x(t) + h_{bin(x,t)}(t) = H$). For any time $t \leq Hn - n + 1$, $w_t \leq f_t$.*

PROOF. Switches only occur under Rules 3 and 4. Whenever the algorithm switches under these rules, at least one ball becomes exact H and moves to E. Since a ball in E never leaves its bin, we deduce that $w_t \leq f_t$. □

The following lemma essentially states that the number of requests already served is $-w_t + \sum_i h_i(t)$.

LEMMA 5.1.3. *For any time $t \leq Hn - n + 1$,*

$$(Hn - n) - \sum_{x=1}^{n} r_x(t) = -w_t + \sum_{i=1}^{n} h_i(t) \ .$$

PROOF. On one hand, the total number of requests already served is the overall number minus the ones left to be served, whose number is $\sum_x r_x(t)$. On the other hand, serving a request costs one erasure if the algorithm does not switch balls, or two if it does, so the total number is $\sum_i h_i(t) - w_t$. □

These two lemmas allow us to prove an important property of the algorithm.

LEMMA 5.1.4. *For any time $t \leq Hn - n + 1$, if there is a dangerous ball then there is a safe ball.*

PROOF. Suppose for contradiction that there is a dangerous ball but no safe ball. Let D_t denote the set of dangerous balls at time t. We have

$$\begin{aligned}
Hn - n &= -w_t + \sum_{x=1}^{n} r_x(t) + \sum_{i=1}^{n} h_i(t) \\
&= -w_t + \sum_{x=1}^{n} \left(r_x(t) + h_{\text{bin}(x,t)}(t) \right) \\
&\geq -f_t + \sum_{x=1}^{n} \left(r_x(t) + h_{\text{bin}(x,t)}(t) \right) \\
&= -f_t + \sum_{x \in D_t} \left(r_x(t) + h_{\text{bin}(x,t)}(t) \right) + H f_t + (H-1)\left(n - |D_t| - f_t\right) \\
&= \sum_{x \in D_t} \left(r_x(t) + h_{\text{bin}(x,t)}(t) \right) + (H-1)\left(n - |D_t|\right) \\
&\geq H\,|D_t| + (H-1)\left(n - |D_t|\right) \\
&= Hn - n + |D_t| \\
&> Hn - n \ ,
\end{aligned}$$

a contradiction. The first line follows from the previous lemma. The second line follows from renaming the bins in the first equation, and the next inequality follows from Lemma 5.1.2. In line 4 we partitioned the balls into dangerous ones, strongly exact ones, and weakly exact ones (under our supposition there are no safe balls). The second inequality (lines 5–6) follows from the fact that for a dangerous ball x, we have $r_x(t) + h_{\text{bin}(x,t)}(t) \geq H$. □

The last lemma before the main result of the chapter states a crucial invariant of the algorithm. We state the lemma and use it, along with the previous lemmas, to prove the main result. Only then we prove the lemma, whose proof is rather technical and long.

LEMMA 5.1.5. *Let x be a safe ball at time $t \leq Hn - n + 1$ and let y be a dangerous ball at time t. Then*

$$r_x(t) + h_{bin(y,t)}(t) \quad \leq \quad H - 1$$

and

$$r_y(t) + h_{bin(x,t)}(t) \quad \geq \quad H \ .$$

THEOREM 5.1.6. *The wear $h_i(t)$ of any bin i at time $t = Hn - n + 1$ is at most H.*

PROOF. If all the balls at time $t = Hn - n + 1$ are safe or exact, then the theorem holds (since we assumed that all the bins are full at all times). We show that this is indeed the case and that there are no dangerous balls. Suppose for contradiction that y is dangerous at time t. By Lemma 5.1.4, there exists a ball x that is safe at time t.

At time $t = Hn - n + 1$ the number of remaining requests $r_x(t)$ for ball x must be zero. Therefore, $r_x(t) + h_{\text{bin}(x,t)}(t) = h_{\text{bin}(x,t)}(t)$ (exactly the same holds for y). Since y is dangerous,

$$r_y(t) + h_{\text{bin}(y,t)}(t) = h_{\text{bin}(y,t)}(t) > H \ .$$

By Lemma 5.1.5,

$$r_x(t) + h_{\text{bin}(y,t)}(t) \ \leq \ H - 1 \ ,$$

but this is a contradiction, since

$$r_x(t) + h_{\text{bin}(y,t)}(t) = h_{\text{bin}(y,t)}(t) > H \ .$$

Therefore, a dangerous ball y cannot exist at time t and the theorem holds. □

All that remains now is to prove Lemma 5.1.5.

PROOF. By induction on t. For $t = 1$ we have $h_i(1) = 0$ for every bin i, and because x is safe and y is dangerous, we also have $r_x(1) < H - 1$ and $r_y(1) > H$, the lemma holds for $t = 1$.

Let us assume correctness at time $t - 1$ and prove that the lemma holds at time t. We denote by D_t the set of dangerous balls at time t, by S_t the set of safe balls at time t, and by E_t the set of exact balls at time t. We split the analysis into cases according to the way σ_{t-1} was served.

- Suppose that σ_{t-1} it was served by rule 1. By lemma 5.1.1, σ_{t-1} is exact. This implies that all the properties of safe and dangerous balls and their bins remain the same as they were at time $t - 1$, so the lemma holds.
- Suppose that σ_{t-1} it was served by rule 2. It is easy to see that $S_t = S_{t-1}$ and $D_t = D_{t-1}$ (because leaving a ball in its bin does not change its safety property). Therefore we know x was safe and y was dangerous at time $t - 1$. We know that
  - $r_x(t) \leq r_x(t-1)$
  - $h_{\text{bin}(x,t)}(t) \geq h_{\text{bin}(x,t-1)}(t-1)$
  - $r_y(t) = r_y(t-1)$ (because y is dangerous but σ_{t-1} is safe)
  - $h_{\text{bin}(y,t)}(t) = h_{\text{bin}(y,t-1)}(t-1)$ (for the same reason)

  By the induction assumption we have $r_x(t-1) + h_{\text{bin}(y,t-1)}(t-1) \leq H - 1$. Therefore, $r_x(t) + h_{\text{bin}(y,t)}(t) \leq H - 1$. We also have $r_y(t) + h_{\text{bin}(x,t)}(t) = r_y(t-1) + h_{\text{bin}(x,t-1)}(t-1) \geq H$ (the inequality is by the inductive assumption).
- Suppose that σ_{t-1} was served by rule 3. We denote by z the ball that the rule switches with σ_t. By the rule, at time $t-1$ the ball z is safe and the ball σ_{t-1} is dangerous (if σ_t was exact or safe, it would have been served by rule 1 or rule 2). By the inductive assumption,

$r_z(t-1) + h_{\text{bin}(\sigma_{t-1},t-1)}(t-1) \leq H-1$. Therefore,

$$r_z(t) + h_{\text{bin}(z,t)}(t) - 1 = r_z(t-1) + \left(h_{\text{bin}(\sigma_{t-1},t-1)}(t-1)+1\right) - 1 \leq H-1 \ ,$$

so $r_z(t) + h_{\text{bin}(z,t)}(t) \leq H$. If $r_z(t) + h_{\text{bin}(z,t)}(t) \in \{H, H-1\}$ then $z \in E_t$. This implies that the lemma holds at time t, because only z and σ_{t-1} change their state, and both are in E_t. We now analyze the remaining case, $r_z(t)+h_{\text{bin}(z,t)}(t) < H-1$. Hence, we have $D_t \subset D_{t-1}$, which implies that y is dangerous at time $t-1$. If $x \neq z$, then the lemma trivially holds at time t, since nothing changed in x's or y's bins. Otherwise, $x = z$. We have $r_{\sigma_{t-1}}(t-1)+h_{\text{bin}(x,t-1)}(t-1) = H$, and by induction $r_y(t-1)+h_{\text{bin}(x,t-1)}(t-1) \geq H$. Therefore, $r_{\sigma_{t-1}}(t-1) \leq r_y(t-1)$. Since σ_{t-1} is dangerous at time $t-1$, we have $r_{\sigma_{t-1}}(t-1) + h_{\text{bin}(\sigma_{t-1},t-1)}(t-1) > H$. Therefore, $r_y(t)+h_{\text{bin}(\sigma_{t-1},t-1)}(t)-1 > H$ which implies that $r_y(t) + h_{\text{bin}(\sigma_{t-1},t-1)}(t) > H+1$. Since $\text{bin}(\sigma_{t-1}, t-1) = \text{bin}(x,t)$, we have $r_y(t) + h_{\text{bin}(x,t)}(t) > H+1$. We also know by the induction assumption that $r_x(t-1) + h_{\text{bin}(y,t)}(t-1) \leq H-1$, which implies $r_x(t) + h_{\text{bin}(y,t)}(t) \leq H-1$ (both r_x and $h_{\text{bin}(y,t)}$ did not change when σ_{t-1} was served).

- Suppose that σ_t was served by rule 4: it was switched with some other ball z, which was safe at time $t-1$ and became exact at time t. By induction, $r_{\sigma_{t-1}}(t-1) + h_{\text{bin}(z,t-1)}(t-1) \geq H$. Therefore, $\left(r_{\sigma_{t-1}}(t)+1\right) + \left(h_{\text{bin}(\sigma_{t-1},t)}(t)-1\right) \geq H$. Therefore, σ_{t-1} is dangerous at time t. Therefore, we have $S_t \subset S_{t-1}$, so x is safe at time $t-1$. If $y \neq \sigma_{t-1}$ then the lemma trivially holds at time t (since nothing changes in x's or y's bins). Otherwise, $y = \sigma_{t-1}$. Rule 4 ensures that $r_z(t-1) + \left(h_{\text{bin}(y,t-1)}(t-1)+1\right) = H$. Since $\sigma_{t-1} = y$ is dangerous at time $t-1$, we have by induction $r_x(t-1) + h_{\text{bin}(y,t-1)}(t-1) \leq H-1$, so $r_z(t-1) \geq r_x(t-1)$. Since z is safe at time $t-1$, we have $r_z(t-1)+h_{\text{bin}(z,t-1)}(t-1) < H-1$. Since $x \neq y = \sigma_{t-1}$, $r_x(t) = r_x(t-1)$. Therefore,

$$\begin{aligned} r_x(t) + h_{\text{bin}(y,t)}(t) - 1 &= r_x(t-1) + h_{\text{bin}(z,t-1)}(t-1) \\ &\leq r_z(t-1) + h_{\text{bin}(z,t-1)}(t-1) \\ &< H-1 \ , \end{aligned}$$

so $r_x(t)+h_{\text{bin}(y,t)}(t) < H$. Since all of the terms are integers, we have $r_x(t) + h_{\text{bin}(y,t)}(t) \leq H-1$.
To complete the analysis of this rule, we show that $r_y(t)+h_{\text{bin}(x,t)}(t) \geq H$. We show by contradiction that $r_y(t-1) + h_{\text{bin}(x,t-1)}(t-1) \neq H$. If the two sides are equal, then the algorithm would serve $y = \sigma_{t-1}$ using rule 3, which it did not. By induction, we have $r_y(t-1) + h_{\text{bin}(x,t-1)}(t-1) \geq H$. Therefore, $r_y(t-1) + h_{\text{bin}(x,t-1)}(t-1) > H$. This

implies $r_y(t) + 1 + h_{\text{bin}(x,t)}(t) > H$. Since all terms here are integers we get $r_y(t) + h_{\text{bin}(x,t)}(t) \geq H$.

- Finally, suppose that σ_{t-1} was served by rule 5. It is easy to see that $S_{t-1} = S_t$, and $D_{t-1} = D_t$ (because putting a ball back in its bin doesn't change its safety property). Therefore, x is safe and y is dangerous at time $t-1$. If $y \neq \sigma_{t-1}$ then the lemma trivially holds at time t (since nothing changes in x's or y's bins). Otherwise, $y = \sigma_{t-1}$. By induction, $r_x(t-1) + h_{\text{bin}(y,t-1)}(t-1) \leq H - 1$ and $r_y(t-1) + h_{\text{bin}(x,t-1)}(t-1) \geq H$. We have $r_x(t-1) + h_{\text{bin}(y,t-1)}(t-1) \neq H-1$ since the algorithm did not apply Rule 4. We also know that $r_y(t-1) + h_{\text{bin}(x,t-1)}(t-1) \neq H$ because the algorithm did not apply Rule 3. Therefore, $r_x(t-1)+h_{\text{bin}(y,t-1)}(t-1) = r_x(t)+\big(h_{\text{bin}(y,t)}(t) - 1\big) < H-1$ and $r_y(t-1) + h_{\text{bin}(x,t-1)}(t-1) = (r_y(t)+1) + h_{\text{bin}(x,t)}(t) > H$. Since all the terms are integers, we get $r_x(t) + h_{\text{bin}(y,t)}(t) \leq H-1$ and $r_y(t) + h_{\text{bin}(x,t)}(t) \geq H$.

□

5.2. An Atomic Variant

We now describe an atomic variant of the previous algorithm. Without any empty bins, there is no way to serve a request atomically. Therefore, in the atomic case, we assume that $n > m$. We use a reduction to the non-atomic problem and show that the atomic offline algorithm still achieves $\ell_{\mathsf{off}}(\sigma) \geq Hn - n$. Again, for simplicity we show an algorithm for the case $n = m + 1$, which gives a weaker lower bound, but only by a negligible additive term (if there is more than one empty bin, we assume that the other empty bins contain dummy balls that are never requested). To keep the mapping between balls and bins bijective, in an atomic algorithm we define $\text{bin}(n,t)$ to be the empty bin and $\text{ball}(y,t) = n$ if y is the empty bin.

Let $\mathcal{T}$ denote the following trivial algorithm for the atomic variant: it serves σ_t by putting σ_t in the single empty bin and erases $\text{bin}(\sigma_t, t)$. Let $\mathcal{N}$ denote the non-atomic algorithm from Section 5.1. Let $\mathcal{A}$ denote the atomic algorithm that we describe below.

The algorithm $\mathcal{A}$ works as follows. Its input is n and a request sequence σ. The algorithm first simulates $\mathcal{T}$ on σ. It uses the behavior of $\mathcal{T}$ on σ to generate another sequence η. It then simulates $\mathcal{N}$ on η. Finally, the actions of $\mathcal{N}$ on η are transformed into atomic actions on σ.

The generation of η is simple: $\eta_t = \text{bin}_{\mathcal{T}}(\sigma_t, t)$, where $\text{bin}_{\mathcal{T}}(\sigma_t, t)$ denotes the bin that contains ball σ_t just before algorithm $\mathcal{T}$ serves the tth request. Note that η refers to balls $1, 2, \ldots, m, m+1 = n$ even though σ only refers to m balls.

The transformation of $\mathcal{N}$'s actions on η into actions of $\mathcal{A}$ on σ works as follows:

- If $\mathcal{N}$ leaves η_t in its bin, $\mathcal{A}$ puts σ_t in the empty bin at time t.
- If $\mathcal{N}$ switches η_t with some other ball x, $\mathcal{A}$ first puts σ_t in the empty bin at time t, and then moves the ball from $\text{bin}_{\mathcal{N}}(x,t)$ to $\text{bin}_{\mathcal{A}}(\sigma_t,t)$ (which is now empty). We will show later that this is a meaningful move, in the sense that $\text{bin}_{\mathcal{N}}(x,t) \neq \text{bin}_{\mathcal{A}}(\sigma_t,t)$.

There is another (perhaps more intuitive) way to define $\mathcal{A}$. We could define it as the same algorithm described in the previous section, but replacing the concept of balls with ball chains. Consider how $\mathcal{T}$ behaves on σ. It moves σ_t out of its bin i; in the next step, it puts σ_{t+1} in i (i.e. $\eta_t = i$). Therefore, the next time the wear of i will increase is when for some t_2, $\sigma_{t_2} = \sigma_{t+1}$ (and then $\eta_{t_2} = i$). We can see that the balls requested by η are chains of balls in σ. The chain in σ that corresponds to ball i in η is $\sigma_{t_1}...\sigma_{t_p}$, where $\sigma_{t_1} = i$, and $\sigma_{t_{i+1}} = \sigma_{t_i+1}$. There are n such chains. The algorithm $\mathcal{A}$ simply treats these chains like $\mathcal{N}$ treats balls, with an appropriate mapping of the two actions types (leaving a ball in its bin or switching it).

To analyze the algorithm, we define a sequence of mappings between balls in $\mathcal{N}$ and $\mathcal{A}$.

DEFINITION 5.2.1. We denote the mapping function $g_t : \{1,2,\dots n\} \to \{1,2,\dots n\}$ at time t using the recurrence

$$\begin{aligned} g_1(x) &= x \\ g_{t+1}(x) &= \begin{cases} g_t(n) & x = \sigma_t \\ g_t(\sigma_t) & x = n \\ g_t(x) & \text{otherwise} \end{cases} \end{aligned}$$

That is, $g_t(x)$ is the ball from the execution of $\mathcal{N}$ that matches the ball x from the execution of $\mathcal{A}$ at time t. We shall show below that this mapping function extends the translation from σ to η in the sense that $g_t(\eta_t) = \sigma_t$.

LEMMA 5.2.2. *For every time $t \leq Hn - n + 1$ and for each ball x, $g_t(x) = \textit{bin}_{\mathcal{T}}(x,t)$.*

PROOF. By induction on t. For $t = 1$ the lemma is trivial. Let us assume correctness for time t and prove it for $t+1$. For $x \neq \sigma_t, n$ we have $\text{bin}_{\mathcal{T}}(x,t+1) = \text{bin}_{\mathcal{T}}(x,t) = g_t(x) = g_{t+1}(x)$. Also, $\text{bin}_{\mathcal{T}}(\sigma_t,t+1) = \text{bin}_{\mathcal{T}}(n,t) = g_t(n) = g_{t+1}(\sigma_t)$ and $\text{bin}_{\mathcal{T}}(n,t+1) = \text{bin}_{\mathcal{T}}(\sigma_t,t) = g_t(\sigma_t) = g_{t+1}(n)$. □

COROLLARY 5.2.3. *For every time $t \leq Hn - n + 1$ we have $\eta_t = g_t(\sigma_t)$*

LEMMA 5.2.4. *For every time $t \leq Hn - n + 1$, $\textit{bin}_{\mathcal{A}}(x,t) = \textit{bin}_{\mathcal{N}}(g_t(x),t)$.*

PROOF. By induction on t. For $t = 1$ the lemma is trivial. For $t > 1$ we split the analysis according to how η_t was served by $\mathcal{N}$: in-place or switching.

If η_t did not cause a switch, we know that for $x \neq \sigma_t, n$, $\text{bin}_{\mathcal{A}}(x,t+1) = \text{bin}_{\mathcal{A}}(x,t)$ and $g_{t+1}(x) = g_t(x)$. Also, we know that for any ball z $\text{bin}_{\mathcal{N}}(z,t+$

$1) = \text{bin}_{\mathcal{N}}(z,t)$. Therefore for $x \neq \sigma_t, n$ the lemma follows from the induction assumption. For $x = \sigma_t$ we have

$$\begin{aligned}\text{bin}_{\mathcal{A}}(\sigma_t, t+1) &= \text{bin}_{\mathcal{A}}(n,t) \\ &= \text{bin}_{\mathcal{N}}(g_t(n),t) \\ &= \text{bin}_{\mathcal{N}}(g_{t+1}(\sigma_t),t) \\ &= \text{bin}_{\mathcal{N}}(g_{t+1}(\sigma_t),t+1)\end{aligned}$$

The proof for the case $x = n$ is similar.

If η_t was switched with another ball y, the analysis is more complex. We have $\text{bin}_{\mathcal{N}}(y,t+1) = \text{bin}_{\mathcal{N}}(\eta_t,t)$ and $\text{bin}_{\mathcal{N}}(\eta_t,t+1) = \text{bin}_{\mathcal{N}}(y,t)$. The atomic algorithm $\mathcal{A}$ serves σ_t by moving the balls in three bins, $\text{bin}_{\mathcal{A}}(\sigma_t,t)$, $\text{bin}_{\mathcal{A}}(n,t)$ and $\text{bin}_{\mathcal{N}}(y,t)$. There must be some v such that $g_t(v) = y$ (since g_t is bijective). For $x \neq \sigma_t, n, v$ we clearly have $\text{bin}_{\mathcal{A}}(x,t+1) = \text{bin}_{\mathcal{N}}(g_{t+1}(\sigma_t), t+1)$.

We now analyse the balls σ_t, n, and v that $\mathcal{A}$ moves in step t, first under the assumption that $v \neq n$ and then under the opposite assumption. Assume that $v \neq n$. We first show the lemma for ball v. Clearly, $\text{bin}_{\mathcal{A}}(\sigma_t,t) = \text{bin}_{\mathcal{A}}(v,t+1)$. By the induction assumption and Corollary 5.2.3 we know that $\text{bin}_{\mathcal{A}}(\sigma_t,t) = \text{bin}_{\mathcal{N}}(\eta_t,t)$. Therefore,

$$\text{bin}_{\mathcal{A}}(v,t+1) = \text{bin}_{\mathcal{N}}(y,t+1) = \text{bin}_{\mathcal{N}}(g_t(v),t+1) = \text{bin}_{\mathcal{N}}(g_{t+1}(v),t+1)\ .$$

The last equality follows from the assumption that $v \neq n$ and from the fact that $v \neq \sigma_t$, which is true since $g_t(v) = y \neq \eta_t = g_t(\sigma_t)$ (this also shows that the definition of the algorithm is valid). We now move to the second ball, σ_t. For it, it does not matter whether η_t is switched or not: it is always moved to the empty bin. The third ball is n. We know that $\text{bin}_{\mathcal{A}}(n,t+1) = \text{bin}_{\mathcal{A}}(v,t) = \text{bin}_{\mathcal{N}}(y,t)$. Because $\mathcal{N}$ switched η_t with y, $\text{bin}_{\mathcal{N}}(y,t) = \text{bin}_{\mathcal{N}}(\eta_t,t+1)$. We also know that $\eta_t = g_t(\sigma_t) = g_{t+1}(n)$. The last three identities imply that $\text{bin}_{\mathcal{A}}(n,t+1) = \text{bin}_{\mathcal{N}}(g_{t+1}(n),t+1)$.

We now assume that $v = n$. In this case, $\mathcal{A}$ first moves σ_t to the empty bin (the one holding the pseudo ball n), and then moves σ_t back to its original bin. This implies that $\text{bin}_{\mathcal{A}}(\sigma_t,t) = \text{bin}_{\mathcal{A}}(\sigma_t,t+1)$ and $\text{bin}_{\mathcal{A}}(n,t) = \text{bin}_{\mathcal{A}}(n,t+1)$. By the induction assumption we have $\text{bin}_{\mathcal{A}}(\sigma_t,t) = \text{bin}_{\mathcal{N}}(\eta_t,t)$. Moreover, we know that $\text{bin}_{\mathcal{N}}(y,t+1) = \text{bin}_{\mathcal{N}}(\eta_t,t)$. We also know that $y = g_t(n) = g_{t+1}(\sigma_t)$, so

$$\text{bin}_{\mathcal{A}}(\sigma_t,t+1) = \text{bin}_{\mathcal{A}}(\sigma_t,t) = \text{bin}_{\mathcal{N}}(\eta_t,t) = \text{bin}_{\mathcal{N}}(y,t+1) = \text{bin}_{\mathcal{N}}(g_{t+1}(\sigma_t),t+1)\ .$$

For the ball n, we know that $\text{bin}_{\mathcal{A}}(n,t+1) = \text{bin}_{\mathcal{A}}(n,t) = \text{bin}_{\mathcal{N}}(y,t)$. Because $\mathcal{N}$ switched η_t with y, $\text{bin}_{\mathcal{N}}(y,t) = \text{bin}_{\mathcal{N}}(\eta_t,t+1)$. Since $\eta_t = g_t(\sigma_t) = g_{t+1}(n)$, the claim holds.

This concludes the proof of the lemma. □

The next lemma shows that the correspondence between σ and η ensures that the wear of individual bins in the atomic algorithm is exactly the same wear as in the non-atomic one.

LEMMA 5.2.5. *For each time $t \leq Hn - n + 1$ and for each bin i, $h_i^{\mathcal{A}}(t) = h_i^{\mathcal{N}}(t)$, where the superscript denotes the algorithm that is used.*

PROOF. By induction on t. For $t = 1$ the lemma is trivial. Assume the lemma holds up to t. By the previous lemma we have that $\text{bin}_{\mathcal{A}}(\sigma_t, t) = \text{bin}_{\mathcal{N}}(\eta_t, t) = i$. If η_t was not switched, then in both executions, only bin i increases its wear. If η_t was switched with some other ball y, then only bins i and $\text{bin}_{\mathcal{N}}(y, t)$ increase their wear in both executions. Since the wear was equal on all bins at time t, it is equal at time $t + 1$. □

THEOREM 5.2.6. *The wear $h_i^{\mathcal{A}}(t)$ of any bin i at time $t = Hn - n + 1$ is at most H.*

PROOF. This follows from the previous lemma, and from Theorem 5.1.6. □

The transformation of a non-atomic algorithm to an atomic one that we defined here is generally useful.

THEOREM 5.2.7. *Let $\mathcal{N}$ be a non-atomic algorithm for n balls and n bins that serves a request sequence η by either putting η_t back in its bin or by switching η_t with some other ball. We can derive from $\mathcal{N}$ an atomic algorithm $\mathcal{A}$ for $n - 1$ balls and n bins. For any request sequence σ that $\mathcal{A}$ serves, there is another sequence η such that $h_i^{\mathcal{A}(\sigma)}(t) = h_i^{\mathcal{N}(\eta)}(t)$ for all i and for all t. If $\mathcal{N}$ is online then $\mathcal{A}$ is online.*

CHAPTER 6

The Deterministic Online Problem

The deterministic online case is relatively easy to analyze. We analyze a simple algorithm and prove that it is optimal by showing a simple lower bound. The algorithm serves a request for ball x by putting x in the least worn out empty bin (excluding the bin it was taken out of). The algorithm is clearly atomic.

THEOREM 6.0.8. *The wear of any bin after* $(n-m+1)H$ *requests is at most* H.

PROOF. By induction on H. For $H = 0$, the lemma trivially holds. Assume the theorem holds for $H-1$. Let $t = (n-m+1)(H-1)+1$. By the induction assumption we know that at time t (that is, before request t is served), all the wear of bins is at most $H-1$. Therefore, at time t, there are $n-m$ empty bins with wear at most $H-1$. Hence, the next $n-m$ requests will be served by putting balls into bins with wear at most $H-1$. During the period in which these requests are served, the wear of some bins may raise to H, but they will be left empty.

Therefore, at this point in time, all the nonempty bins have wear at most $H-1$. This implies that we can serve at least one more request without raising the wear of any bin to $H+1$.

We have shown that we can serve $(n-m)+1$ requests without raising the overall wear to $H+1$ or higher, which proves the inductive claim and the theorem. □

COROLLARY 6.0.9. *The competitive ratio of the algorithm is at least*

$$\frac{n-m+1}{n+\frac{n-m}{H}}\ .$$

With a single extra bin ($n-m=1$) and large H the ratio is roughly $2/n$. For a large H and a large fraction of extra bins, say $m = n/2$, the ratio is roughly $1/2$.

THEOREM 6.0.10. *For every deterministic algorithm* α *(even if* α *is non-atomic) there exists a sequence* σ *such that* $\ell_\alpha(\sigma) \le (n-m+1)H$.

PROOF. Given α, we will generate σ as follows. Initially there are $n-m$ empty bins. Let T denote an arbitrary set of $n-m+1$ bins. In each moment

during the execution of α, at least one of the bins in T is not empty. The sequence σ will always request one of the balls which is in a bin of T. This ensures that after every request t, the total wear

$$\sum_{i \in T} h_i(t)$$

of the bins of T grows by at least 1. This means that the best α can do is to distribute the wear evenly among bins of T, which allows α to serve at most $(n - m + 1)H$ requests. □

Clearly, our deterministic algorithm is optimal in the worst-case sense.

COROLLARY 6.0.11. *The competitive ratio of any deterministic algorithm is at most*

$$\frac{n - m + 1}{n - \frac{n}{H}} .$$

The gap between the lower bound in Corollary 6.0.9 and the upper bound in Corollary 6.0.11 is due to the gap in our estimation of the optimal offline algorithm (in the denominator).

CHAPTER 7

The Randomized Online Unit Problem

We now analyze the non-atomic randomized case when $n = m$ and the atomic randomized case when $n = m + 1$. We view H as function of n. We present a simple randomized online algorithm and show that it is asymptotically optimal (in the sense that its competitive ratio is $1 - o(1)$) when $H/\ln n = \omega(1)$. In practice, H is large, usually at least $100,000$ (in 2006), so this part of the analysis applies to most practical cases. For smaller H (unlikely in practice), the competitive ratio of the algorithm is

$$\frac{n^{-\Theta(1/H)} \ln n}{H} .$$

We also show that for any algorithm there is a sequence such that with high probability the algorithm is unable to serve more than $n^{1-\Theta(1/H)} \ln n$ requests of the sequence (which hints our algorithm is optimal in some sense).

The algorithm that we analyze is not atomic, but the reduction in Section 5.2 can transform it into an atomic algorithm with the same asymptotic competitive ratio. The algorithm that we analyze is essentially Ban's algorithm, which he patented in [**5**]; the algorithm that we analyze corresponds to Claims 2.c, 3, and 4.II in [**5**].

The asymptotics in this chapter are always with respect to a growing n, unless specified otherwise.

The next chapter investigates this randomized algorithm experimentally.

7.1. The Case $\Omega(\frac{1}{\ln \ln n}) \leq \frac{H}{\ln n} \leq O(1)$

We begin the analysis with the range of H that is easier to analyze: smaller H. Our analysis does not apply to very small H, where the prospects for effective wear leveling are small.

The algorithm that we propose and analyze, called $\mathcal{R}_p$, is quite simple, but in the case of small H we can use an even simpler one, called $\mathcal{R}_1$. Given a request for ball x, $\mathcal{R}_1$ puts x in a random bin i chosen uniformly and independently of previous steps. The algorithm puts the ball that was in bin i in the bin in which x was stored. In particular, with probability $1/n$, the algorithm returns x to the bin in which it was stored.

Notice that $\mathcal{R}_1$ almost always performs 2 erasures per request, which is rather inefficient. However, this will not matter much since the competitive ratio will be small (for any algorithm).

The analysis of $\mathcal{R}_1$ relies upon well known results regarding the following *random increment game*. Consider a game of b rounds in which a distinct counters are participating. All the counters are initialized to zero. In each round, a single counter, which is chosen uniformly and independently at random, is incremented. We denote by $\mathrm{ri}(a, b)$ the distribution of the value of the maximum counter in such a game.[1]

The sequence σ can cause an uneven wear in spite of the random choices that the algorithm makes. Suppose that the first two requests in σ are for the same ball. One of the bins will surely be erased twice: the bin into which the algorithm put ball σ_1 when it was serving the first request. Therefore, the bins that the algorithm erases in a particular step are dependent upon the previous random choices that the algorithm made.

To deal with this dependence, we define two sets of n counters, $c_1, \ldots, c_n$ and $d_1, \ldots, d_n$. When $\mathcal{R}_1$ serves σ_t, it moves the ball σ_t from $\mathrm{bin}(\sigma_t, t)$ to $\mathrm{bin}(\sigma_t, t+1)$. To account for that, we increment counters $c_{\mathrm{bin}(\sigma_t,t)}$ and $d_{\mathrm{bin}(\sigma_t,t+1)}$. That is, after σ_t is served, c_i is the wear of bin i, when we only count wear caused by taking the requested ball out of its bin. Similarly, d_i is the wear of bin i when we only count wear caused by taking a ball y out of its bin when y is not requested in that particular step, but rather is chosen randomly by the algorithm. We also define $C^{(t)} = \max_i c_i^{(t)}$, where $c_i^{(t)}$ is the value of c_i after t requests are served. Similarly, $D^{(t)} = \max_i d_i^{(t)}$.

The next lemma shows that the unevenness that a particular σ can cause in the wear is represented completely by dependence between the c's and the d's; within each set of counters, the distributions are completely uniform and independent.

Without loss of generality, we assume that initially each ball is in a random bin (without this assumption, the analysis still holds for a renaming of the bins).

LEMMA 7.1.1. *$C^{(t)} \sim ri(n, t)$ (the distribution of the random variable $C^{(t)}$ is $ri(n, t)$) and $D^{(t)} \sim ri(n, t)$.*

PROOF. The fact that $D^{(t)} \sim \mathrm{ri}(n, t)$ follows from the definition of $\mathcal{R}_1$, since after each request the incremented d_i is chosen uniformly and independently at random. The result concerning $C^{(t)}$ follows from the fact that the conditional distribution of $\mathrm{bin}(\sigma_t, t)$, conditioned on $\mathrm{bin}(\sigma_1, 1), \ldots, \mathrm{bin}(\sigma_{t-1}, t-1)$, is uniform. If this claim holds, the c counter that we increment at

[1] In the literature, the distribution $\mathrm{ri}(a, b)$ is often referred to as the distribution of "balls and bins". We do not use this term in this paper to avoid confusion with balls that represent logical sectors and bins that represent flash erase units.

step t is indeed random and independent of previous random increments. We prove the claim concerning the distribution of $\text{bin}(\sigma_t, t)$ by inductively showing that for all x, $\text{bin}(x,t)$ is uniformly distributed even when conditioned on $\text{bin}(\sigma_1, 1), \ldots, \text{bin}(\sigma_{t-1}, t-1)$. By our assumption on the initial distribution of the balls, the claim holds for $t = 1$. Suppose that the claim is true for $\text{bin}(x, t-1)$ for any x. By the definition of $\mathcal{R}_1$, for any bin i it holds that

$$\Pr\left[\text{bin}(\sigma_{t-1}, t) = i | \text{bin}(\sigma_1, 1), \ldots, \text{bin}(\sigma_{t-1}, t-1)\right] = \frac{1}{n}\,.$$

We now analyze $\Pr\left[\text{bin}(x,t) = i | \text{bin}(\sigma_1, 1), \ldots, \text{bin}(\sigma_{t-1}, t-1)\right]$ for $x \neq \sigma_{t-1}$. The event $\text{bin}(x,t) = i$ occurs when

$$\begin{aligned} e = (&\text{bin}(x, t-1) = i \text{ and } \text{bin}(\sigma_{t-1}, t) \neq i) \\ &\text{or } (\text{bin}(x, t-1) = \text{bin}(\sigma_{t-1}, t) \text{ and } \text{bin}(\sigma_{t-1}, t-1) = i) \end{aligned}$$

occurs. Thus, if $\text{bin}(\sigma_{t-1}, t-1) = i$, then the second part of e occurs, so

$$\begin{aligned} &\Pr\left[\text{bin}(x,t) = i | \text{bin}(\sigma_1, 1), \ldots, \text{bin}(\sigma_{t-1}, t-1) = i\right] \\ &\quad = \Pr\left[\text{bin}(x, t-1) = \text{bin}(\sigma_{t-1}, t) | \text{bin}(\sigma_1, 1), \ldots, \text{bin}(\sigma_{t-1}, t-1) = i\right] \\ &\quad = \frac{1}{n} \end{aligned}$$

(the uniformity here follows from the fact that $\mathcal{R}_1$ puts σ_{t-1} in a random bin).

On the other hand, if $\text{bin}(\sigma_{t-1}, t-1) \neq i$ then

$$\begin{aligned} &\Pr\left[\text{bin}(x,t) = i | \text{bin}(\sigma_1, 1), \ldots, \text{bin}(\sigma_{t-1}, t-1) = j \neq i\right] \\ &\quad = \Pr[\text{bin}(x, t-1) = i \text{ and } \text{bin}(\sigma_{t-1}, t) \neq i \\ &\qquad | \text{bin}(\sigma_1, 1), \ldots, \text{bin}(\sigma_{t-1}, t-1) = j \neq i] \\ &\quad = \frac{1}{n-1} \cdot \frac{n-1}{n} = \frac{1}{n}\,. \end{aligned}$$

□

The following two lemmas regarding the $\text{ri}(a,b)$ distribution are corollaries of Theorem 1 in [**20**].

LEMMA 7.1.2. *Suppose b is a function of a and $\frac{a}{\text{polylog}(a)} \leq b \ll a \ln a$, where the asymptotics are with respect to a growing a. Then*

$$\Pr\left[\textit{ri}(a,b) < \frac{\ln a}{\ln \frac{a \ln a}{b}}\right] = o(1)$$

and

$$\Pr\left[\boldsymbol{ri}(a,b) > 3\frac{\ln a}{\ln \frac{a\ln a}{b}}\right] = o(1) .$$

LEMMA 7.1.3. *For any constant $d > 0$ there exist constants $c_1, c_2 > 0$ such that*

$$\Pr\left[\boldsymbol{ri}(a, c_1 a \ln a) < d \ln a\right] = o(1) .$$

and

$$\Pr\left[\boldsymbol{ri}(a, c_2 a \ln a) > d \ln a\right] = o(1) ,$$

where the asymptotics are with respect to a growing a.

We are now ready to prove the main result regarding $\mathcal{R}_1$.

THEOREM 7.1.4. *For H in the range*

$$\Omega(\frac{1}{\ln\ln n}) \leq \frac{H}{\ln n} = O(1) ,$$

there exists a constant $c > 0$ such that for every request sequence σ,

$$\Pr\left[\ell_{\mathcal{R}_1}(\sigma) < n^{1-\frac{c}{H}} \ln n\right] = o(1) .$$

PROOF. We analyze the state of the bins after t requests have been served. We split the analysis into two cases, depending on the relationship between H and n.

If $\frac{H}{\ln n} = o(1)$ then we set $c = 6$ and $t = n^{1-\frac{6}{H}} \ln n$. Since $\frac{n}{\text{polylog}(n)} \leq n^{1-\frac{6}{H}} \ln n \ll n \ln n$, by Lemma 7.1.2 we have

$$\Pr\left[C^{(t)} > 3\frac{\ln n}{\ln \frac{n\ln n}{t}}\right] = \Pr\left[D^{(t)} > 3\frac{\ln n}{\ln \frac{n\ln n}{t}}\right] = o(1) .$$

Since

$$\frac{3\ln n}{\ln \frac{n\ln n}{t}} = \frac{3\ln n}{\ln \frac{n\ln n}{n^{1-\frac{6}{H}}\ln n}} = \frac{3\ln n}{\ln n^{6/H}} = \frac{H}{2} ,$$

we obtain

$$\Pr\left[C^{(t)} > H/2\right] = \Pr\left[D^{(t)} > H/2\right] = o(1) .$$

Therefore

$$\begin{aligned}
\Pr\left[\ell_{\mathcal{R}_1}(\sigma) < t\right] &\leq \Pr\left[C^{(t)} + D^{(t)} > H\right] \\
&\leq \Pr\left[(C^{(t)} > H/2) \text{ or } (D^{(t)} > H/2)\right] \\
&\leq \Pr\left[C^{(t)} > H/2\right] + \Pr\left[D^{(t)} > H/2\right] \\
&= o(1) .
\end{aligned}$$

We now analyze the case in $\frac{H}{\ln n} \in O(1)$ but $\frac{H}{\ln n} \notin o(1)$: the case $\frac{H}{\ln n} = \Theta(1)$. For simplicity, we assume that $H = d \ln n$ for some constant d; the general case $\frac{H}{\ln n} = \Theta(1)$ follows easily. We observe that for any constant $c' > 0$ we

have $n^{1-\frac{c'}{H}} \ln n = \Theta(n \ln n)$. By Lemma 7.1.3 there exists a constant $c > 0$ such that for $t = n^{1-\frac{c}{H}} \ln n$ we have

$$\Pr\left[C^{(t)} > (d/2)\ln n\right] \;=\; \Pr\left[D^{(t)} > (d/2)\ln n\right] = o(1)\ .$$

Observing that $(d/2)\ln n = H/2$ we get $\Pr\left[C^{(t)} > H/2\right] = \Pr\left[D^{(t)} > H/2\right] = o(1)$. Therefore

$$\Pr\left[\ell_{\mathcal{R}_1}(\sigma) < t\right] \leq \Pr\left[C^{(t)} + D^{(t)} > H\right] = o(1)\ .$$

□

To bound the competitive ratio, we need to bound $\mathbf{E}\left[\ell_{\mathcal{R}_1}(\sigma)\right]$. By Theorem 7.1.4 we obtain

$$\begin{aligned}\mathbf{E}\left[\ell_{\mathcal{R}_1}(\sigma)\right] &\geq \Pr\left[\ell_{\mathcal{R}_1}(\sigma) \geq n^{1-O(1/H)}\ln n\right] \cdot n^{1-O(1/H)}\ln n \\ &= (1-o(1))n^{1-O(1/H)}\ln n = n^{1-O(1/H)}\ln n\ .\end{aligned}$$

In fact, Theorem 7.1.4 implies a stronger statement than just a bound on the expectation. It shows that the random variable $\ell_{\mathcal{R}_1}(\sigma)$ is highly concentrated above $n^{1-O\left(\frac{1}{H}\right)}\ln n$. Therefore, $\mathcal{R}_1$ will be able to serve any sequence of length $n^{1-O\left(\frac{1}{H}\right)}\ln n$ with high probability.

COROLLARY 7.1.5. *The competitive ratio of the algorithm is at least*

$$\frac{n^{-O(1/H)}\ln n}{H}\ .$$

To solve the atomic problem, we observe that our algorithm satisfies the conditions of Theorem 5.2.7. Therefore we can easily derive an atomic algorithm with the same competitive ratio.

We show that $\mathcal{R}_1$ is asymptotically optimal by showing a matching lower bound. In order for the bound to be applicable to atomic algorithms as well, we prove the bound for a more general setting that allows a constant number of empty bins.

THEOREM 7.1.6. *For every randomized online algorithm α (even if α is non-atomic) and for every constant e such that $n = m + e$, there exists a sequence σ and a constant c such that* $\Pr\left[\ell_\alpha(\sigma) < n^{1-\frac{c}{H}}\ln n\right] \geq 1 - o(1)$.

PROOF. Consider a sequence σ such that σ_t is chosen independently uniformly at random. Let $c_1, \ldots, c_m$ be a set of m counters. A counter is incremented whenever α serves a request, as follows. When α serves σ_t, the incremented counter depends upon $\mathrm{bin}(\sigma_t, t)$. If $\mathrm{bin}(\sigma_t, t) \leq m$ then $c_{\mathrm{bin}(\sigma_t,t)}$ is incremented. Otherwise, a random counter c_i is incremented, where i is chosen uniformly from the set of empty bins at time t whose index is at most m. Let $C^{(t)} = \max_i c_i^{(t)}$, where $c_i^{(t)}$ is the value of c_i after t requests are served. The construction of σ guarantees that $C^{(t)} \sim \mathbf{ri}(m, t)$. Let $h'_i(t)$ denote the wear of bin i, when we only consider wear caused by

a requested ball x taken out of its bin. In view of the fact that c_i is incremented when either h'_i or one of $h'_{m+1}, \ldots, h'_{m+e}$ is incremented, it follows that $c_i^{(t)} \leq h'_i(t) + \sum_{j=m+1}^{m+e} h'_j(t)$.

We analyze the state of the bins after t requests have been served. If $\frac{H}{\ln n} = o(1)$ then we set $t = n^{1-\frac{1}{(e+1)H}} \ln n$. Since $\frac{n}{\text{polylog}(n)} \leq n^{1-\frac{1}{(e+1)H}} \ln n \ll n \ln n$, by Lemma 7.1.2 we have

$$\Pr\left[C^{(t)} > \frac{\ln n}{\ln \frac{n \ln n}{t}}\right] \;=\; 1 - o(1)\;,$$

where the probability is taken over the both α's random choices, the sequence σ, and the randomly selected c_i's. Observing that

$$\frac{\ln n}{\ln \frac{n \ln n}{t}} \;=\; \frac{\ln n}{\ln \frac{n \ln n}{n^{1-\frac{1}{(e+1)H}} \ln n}} = \frac{\ln n}{\ln n^{1/(e+1)H}} = (e+1)H$$

we get $\Pr\left[C^{(t)} > (e+1)H\right] = 1 - o(1)$. Therefore

$$\begin{aligned}
\Pr\left[\ell_\alpha(\sigma) < t\right] &\geq \Pr\left[\exists i\; h'_i(t) > H\right] \\
&\geq \Pr\left[\exists i\; h'_i(t) + \sum_{j=m+1}^{m+e} h'_j(t) > (e+1)H\right] \\
&\geq \Pr\left[\exists i\; c_i^{(t)} > (e+1)H\right] \\
&= \Pr\left[C^{(t)} > (e+1)H\right] = 1 - o(1)\;.
\end{aligned}$$

We now analyze the case of larger H. Suppose then that $H = d \ln n$ for some constant d. Observe that for any constant c' it holds that $n^{1-\frac{c'}{(e+1)H}} \ln n = \Theta(n \ln n)$. By Lemma 7.1.3 there is a constant c such that for $t = n^{1-\frac{c}{(e+1)H}} \ln n$:

$$\Pr\left[C^{(t)} > (e+1)d \ln n\right] = 1 - o(1)\;.$$

Since $(e+1)d \ln n = (e+1)H$ it follows that

$$\Pr\left[\ell_\alpha(\sigma) < t\right] \geq \Pr\left[C^{(t)} > (e+1)H\right] = \Pr\left[C^{(t)} > (e+1)d \ln n\right] = 1 - o(1)\;.$$

Thus, in both cases, there exists a sequence σ' such that $\Pr\left[\ell_\alpha(\sigma') < t\right] \geq 1 - o(1)$, for $t = n^{1-\frac{c'}{H}} \ln n$, where c' is some constant. □

7.2. The Case $\frac{H}{\ln n} = \omega(1)$

We now analyze the case of higher endurance, which is much more typical in practice. To achieve competitive ratios that are close to 1, we use a slightly more sophisticated algorithm, which we call $\mathcal{R}_p$. It serves a request to a ball x using the following rules:

- With probability p, put x in a random bin i chosen uniformly and independently, and put the ball that was in i in the bin where x was stored (here, again, the algorithm may return x to the bin in which it was stored).
- Otherwise, put x back in the bin from which it was taken out.

Again, to solve the atomic problem, we observe that our algorithm satisfies the conditions of Theorem 5.2.7. Therefore we can easily derive an atomic algorithm with the same competitive ratio.

We use the following notation in the analysis of $\mathcal{R}_p$ (on an arbitrary request sequence). As in the previous case, we wish to analyze separately the two types of wear on each bin: wear caused by a request and wear caused by a switch. We denote by X_i the wear of bin i that is caused by requesting the ball that is stored in it (X_i does not count wear caused by switching the ball in i with another requested ball). We denote by r_i the number of times that the ball in bin i has been requested and that $\mathcal{R}_p$ decided to switch it with a ball in a random bin. We denote by $G(p)$ the geometric distribution with success probability p, and by $B(n,p)$ the binomial distribution of n events with probability p.

LEMMA 7.2.1. *Let $\{G_j\}$ be a set of independent random variables, $G_j \sim G(p)$ for some probability p. For any $a > 0$,*

$$\Pr[X_i > a] \leq \Pr\left[\sum_{j=0}^{r_i} G_j > a\right] .$$

PROOF. Let

$$G'_j = \begin{cases} G_j & j \leq r_i \\ 0 & \text{otherwise} . \end{cases}$$

Clearly $\sum_{j=0}^{r_i} G_j = \sum_{j=0}^{H} G'_j$.

Let $X_{i,j}$ be the random variable that counts the wear of i that is caused by requesting the ball in i, but only between the jth switch (exclusive) and $j+1$st switch (inclusive) on bin i (counting only switches caused by requesting the ball in i). If $j = r_i$, we take $X_{i,j}$ to be all the wear caused by requesting the ball in i after the jth switch. If $j > r_i$, we take $X_{i,j} = 0$. Clearly $X_i = \sum_{j=0}^{H} X_{i,j}$.

We prove the main claim by defining a set of $H+2$ events with probabilities that rise monotonically from $\Pr[X_i > a]$ to $\Pr\left[\sum_{j=0}^{r_i} G_j > a\right]$. For $-1 \leq s \leq H$, we define

$$Z_s = \sum_{j=0}^{s} X_{i,j} + \sum_{j=s+1}^{H} G'_j > a .$$

At the endpoints of the range of s we have $Z_{-1} = \sum_{j=0}^{r_i} G_j$ and $Z_H = X_i$.

We now show that $\Pr[Z_s > a] \leq \Pr[Z_{s-1} > a]$ for every s. We need to show that the sum has a higher probability of being greater than a when $X_{i,s}$ is replaced by G'_s. Consider the subspace of the probability space that is defined by fixing all the algorithm's decision up to the sth switch on bin i. If in this subspace $r_i < s$ then both $X_{i,s}$ and G'_s are 0. Otherwise, $X_{i,s}$ is distributed in this subspace as a truncated geometric variable (with varying truncation value). That is, there exists a Q such that for all $1 \leq q < Q$, $\Pr[X_{i,s} = q] = (1-p)^{q-1}p$, and $\Pr[X_{i,s} = Q] = (1-p)^Q$. On the other hand, once we know that $r_i \geq s$, G'_s is completely independent of the algorithm's decision, so it is a (non-truncated) geometric variable. Thus, when $X_{i,s}$ is replaced by G'_s there is a higher probability for the sum to be greater than a.

The lemma follows from the fact that $Z_H = X_i$ and that $Z_{-1} = \sum_{j=0}^{r_i} G_j$. □

LEMMA 7.2.2. *For any bin i, after T requests are served it holds that $r_i \sim B(T, p/n)$*

PROOF. Denote by $E_{t,i}$ the indicator for the event that when the tth request was served, the ball σ_t was in bin i *and* a switch occurred. Clearly $\Pr[E_{t,i} = 1] = \frac{p}{n}$ and $r_i = \sum_{t=1}^{T} E_{t,i}$. We are only left to show that the indicators $\{E_{t,i}\}_t$ are independent (for a fixed i). It is easy to see that

$$\Pr[E_{t,i} \mid E_{1,i}, \ldots, E_{t-1,i}] \;=\; p \cdot \Pr[\sigma_t \text{ was in bin } i \text{ at time } t \mid E_{1,i}, \ldots, E_{t-1,i}] \ .$$

We claim that the distribution of the balls in bin i is uniform at any time t, regardless of the values of $E_{1,i}, \ldots, E_{t-1,i}$. This follows by induction on t. For $t = 1$ this is true since initially the balls are arranged randomly within the bins. Suppose the claim is true up until time $t-1$. Clearly, if $E_{t-1,i} = 1$ then the ball in bin i at time t is chosen uniformly from all the balls (for any values of $E_{1,i}, \ldots, E_{t-2,i}$). That is, for any ball x, it holds that

$$\Pr[x \text{ was in bin } i \text{ at time } t \mid E_{1,i}, \ldots, E_{t-2,i}, E_{t-1,i} = 1] = \frac{1}{n} \ .$$

Also, by induction we know that given $E_{1,i}, \ldots, E_{t-2,i}$ the distribution of the ball in bin i at time $t-1$ is uniform. But this also means that the ball in bin i at time t is uniform (since no ball can have a higher probability than the others). That is, for any ball x, it holds that

$$\Pr[x \text{ was in bin } i \text{ at time } t \mid E_{1,i}, \ldots, E_{t-2,i}] = \frac{1}{n} \ .$$

From the last two equations it follows that

$$\Pr[x \text{ was in bin } i \text{ at time } t \mid E_{1,i}, \ldots, E_{t-2,i}, E_{t-1,i} = 0] = \frac{1}{n} \ .$$

Thus, the distribution of the balls in bin i is uniform at any time t, regardless of the values of $E_{1,i}, \ldots, E_{t-1,i}$. Therefore, $\Pr\left[E_{t,i} \mid E_{1,i}, \ldots, E_{t-1,i}\right] = p/n$, which implies the indicators $\{E_{t,i}\}_t$ are independent. □

The next lemma is a deviation bound for the sum of geometric random variables.

LEMMA 7.2.3. *Let $A = \sum_{i=1}^{n} A_i$, where for each i, $A_i \sim G(p)$, and the A_i's are independent. Then, for any $\delta > 0$ it holds that*

$$\Pr\left[\sum_{i=1}^{n} A_i > (1+\delta)\frac{n}{p}\right] \leq e^{-\frac{\delta^2 n}{2(1+\delta)}} .$$

PROOF. We can view the distribution of the sum of independent geometric random variables as follows. We preform an infinite number of independent trials, each with success probability p. The value of the sum of n geometric variables is distributed exactly as the number of trials until n success are observed. For the sum to attain a value greater than a, the first a trials must contain less than n successes. Since the number of successes in the first a trials has distribution $B(a,p)$ we get

$$\Pr\left[\sum_{i=1}^{n} A_i > a\right] = \Pr\left[B(a,p) < n\right] .$$

Now, we can use Chernoff bound to derive

$$\begin{aligned}
\Pr\left[\sum_{i=1}^{n} A_i > (1+\delta)\frac{n}{p}\right] &= \Pr\left[B\left((1+\delta)\frac{n}{p}, p\right) < n\right] \\
&= \Pr\left[B\left((1+\delta)\frac{n}{p}, p\right) < \left(1 - \frac{\delta}{1+\delta}\right) \cdot (1+\delta)\frac{n}{p} \cdot p\right] \\
&\leq e^{-\frac{1}{2}\left(\frac{\delta}{1+\delta}\right)^2 \cdot \frac{(1+\delta)n}{p} \cdot p} = e^{-\frac{\delta^2 n}{2(1+\delta)}} .
\end{aligned}$$

□

The next theorem completes the analysis of $\mathcal{R}_p$. When we view p as a function of n, the theorem states which choice of p is asymptotically optimal.

THEOREM 7.2.4. *When $\frac{H}{\ln n} = \omega(1)$, for any $\left(\frac{\ln n}{H}\right)^{1/3} \ll p \ll 1$ and for every request sequence σ,*

$$\Pr\left[\ell_{\mathcal{R}_p}(\sigma) < nH(1 - o(1))\right] = o(1) .$$

PROOF. We analyze the state of the bins after $T = nH/(1+p)^3$ requests have been served and show that the probability that any of them has wear H or higher is only $o(1)$. When we refer to random variables like X_i, we

mean X_i at time T. Since for all p in the allowed range we have $T = nH(1 - o(1))$, the result follows.

Let $X = \max X_i$. Let Y_i be the wear of bin i when we only consider wear added by putting a requested ball into bin i, and let $Y = \max Y_i$. Clearly the maximum wear is bounded by $X + Y$. Therefore:

$$\begin{aligned}
\Pr\left[\ell_{\mathcal{R}_p}(\sigma) < T\right] &\leq \Pr\left[X + Y > H\right] \\
&\leq \Pr\left[\left(X > \frac{H}{1+p}\right) \text{ or } \left(Y > \frac{pH}{1+p}\right)\right] \\
&\leq \Pr\left[X > \frac{H}{1+p}\right] + \Pr\left[Y > \frac{pH}{1+p}\right] .
\end{aligned}$$

We analyze each of the terms individually.

Applying the union bound gives $\Pr\left[X > \frac{H}{1+p}\right] \leq \sum_{i=1}^{n} \Pr\left[X_i > \frac{H}{1+p}\right]$. By Lemma 7.2.1,

$$\Pr\left[X_i > \frac{H}{1+p}\right] \leq \Pr\left[\sum_{j=0}^{r_i} G_j > \frac{H}{1+p}\right]$$

(using the same G_j's as in Lemma 7.2.1). By Lemma 7.2.2 we know that $r_i \sim B(T, p/n)$. Applying Chernoff bound, we obtain

$$\mathbf{Pr}\left[r_i > (1+\delta)\frac{Tp}{n}\right] < e^{-\frac{Tp}{n}\cdot\frac{\delta^2}{3}} .$$

Let B denote the event $r_i \leq (1+\delta)\frac{Tp}{n}$. Therefore,

$$\begin{aligned}
\Pr\left[\sum_{j=0}^{r_i} G_j > \frac{H}{1+p}\right] &= \Pr\left[\sum_{j=0}^{r_i} G_j > \frac{H}{1+p} | B\right] \mathbf{Pr}\left[B\right] + \Pr\left[\sum_{j=0}^{r_i} G_j > \frac{H}{1+p} | \overline{B}\right] \mathbf{Pr}\left[\overline{B}\right] \\
&\leq \Pr\left[\sum_{j=0}^{r_i} G_j > \frac{H}{1+p} | B\right] + \mathbf{Pr}\left[\overline{B}\right] \\
&\leq \Pr\left[\sum_{j=0}^{(1+\delta)\frac{Tp}{n}} G_j > \frac{H}{1+p}\right] + \mathbf{Pr}\left[\overline{B}\right] \\
&\leq \Pr\left[\sum_{j=0}^{(1+\delta)\frac{Tp}{n}} G_j > \frac{H}{1+p}\right] + e^{-\frac{Tp}{n}\cdot\frac{\delta^2}{3}} .
\end{aligned}$$

We now bound the first term in the last line. Lemma 7.2.3 gives

$$\Pr\left[\sum_{j=0}^{(1+\delta)\frac{Tp}{n}} G_j > (1+\delta)\left((1+\delta)\frac{Tp}{n}\right)\frac{1}{p}\right] \leq e^{-(1+\delta)\frac{Tp}{n}\cdot\frac{\delta^2}{2(1+\delta)}} = e^{-\frac{Tp}{n}\cdot\frac{\delta^2}{2}} .$$

Choosing $\delta = p$ we get $(1+\delta)^2\frac{T}{n} = (1+p)^2\frac{nH}{(1+p)^3}\frac{1}{n} = \frac{H}{1+p}$, which implies

$$\Pr\left[\sum_{j=0}^{(1+\delta)\frac{Tp}{n}} G_j > \frac{H}{1+p}\right] \leq e^{-\frac{Tp}{n}\cdot\frac{\delta^2}{2}} .$$

Therefore,

$$\begin{aligned}\Pr\left[X > \frac{H}{1+p}\right] &\leq \sum_{1}^{n}\Pr\left[X_i > \frac{H}{1+p}\right] \\ &\leq n\left(e^{-\frac{Tp}{n}\cdot\frac{\delta^2}{2}} + e^{-\frac{Tp}{n}\cdot\frac{\delta^2}{3}}\right) .\end{aligned}$$

We now prove that Y is unlikely to be large. Let R denote the random variable that counts the total number of ball switches. Let C denote the event $R < (1+\delta)Tp$. Since $R \sim B(T,p)$, by Chernoff bound we have $Pr[\overline{C}] < e^{-Tp\delta^2/3}$. Therefore,

$$\begin{aligned}\Pr\left[Y > \frac{pH}{1+p}\right] &\leq \Pr\left[Y > \frac{pH}{1+p}|C\right] + Pr\left[\overline{C}\right] \\ &\leq \Pr\left[Y > \frac{pH}{1+p}|R = (1+\delta)Tp\right] + Pr\left[\overline{C}\right] \\ &\leq \Pr\left[Y > \frac{pH}{1+p}|R = (1+\delta)Tp\right] + e^{-Tp\delta^2/3} .\end{aligned}$$

The second inequality holds because the probability of a high Y is larger when there are $R = (1+\delta)Tp$ switches than when there are $R < (1+\delta)Tp$ switches.

We now bound $\Pr[Y > pH/(1+p)|R = (1+\delta)Tp]$ using the standard analysis of the random-increment game. Since $Y_i \sim B(R, 1/n)$, Chernoff bound gives

$$\Pr\left[Y_i > (1+\delta)\frac{R}{n} \mid R = R_0\right] < e^{-\frac{R_0}{n}\cdot\frac{\delta^2}{3}} .$$

Therefore, by the union bound we obtain

$$\begin{aligned}\Pr\left[Y > (1+\delta)\frac{R}{n} \mid R = R_0\right] &\leq \sum_{i=1}^{n}\Pr\left[Y_i > (1+\delta)\frac{R}{n} \mid R = R_0\right] \\ &\leq ne^{-\frac{R_0}{n}\cdot\frac{\delta^2}{3}} .\end{aligned}$$

Setting $R_0 = (1+\delta)Tp$ we get

$$\Pr\left[Y > (1+\delta)\frac{R}{n} \mid C\right] \leq ne^{-\frac{(1+\delta)Tp}{n}\cdot\frac{\delta^2}{3}} .$$

Since $\delta = p$, we have

$$(1+\delta)\frac{R_0}{n} = (1+\delta)\frac{(1+\delta)Tp}{n} = (1+p)^2\frac{nHp}{n(1+p)^3} = \frac{pH}{1+p} ,$$

which implies

$$\Pr\left[Y > \frac{pH}{1+p}\right] \leq n\left(e^{-\frac{(1+\delta)Tp}{n}\cdot\frac{\delta^2}{3}} + e^{-Tp\delta^2/3}\right) .$$

Combining the bounds on a large X and on a large Y, we get

$$\begin{aligned}
\Pr\left[\ell_{\mathcal{R}_f}(\sigma) < T\right] &\leq n\left(e^{-\frac{Tp}{n}\cdot\frac{\delta^2}{2}} + e^{-\frac{Tp}{n}\cdot\frac{\delta^2}{3}}\right) + n\left(e^{-\frac{(1+\delta)Tp}{n}\cdot\frac{\delta^2}{3}} + e^{-Tp\delta^2/3}\right) \\
&= n\left(e^{-Hp^3/2(1+p)^3} + e^{-Hp^3/3(1+p)^3} + e^{-Hp^3/3(1+p)^2} + e^{-nHp^3/3(1+p)^3}\right) \\
&\leq n\left(e^{-p^3H/16} + e^{-Hp^3/24} + e^{-Hp^3/12} + e^{-Hp^3/24}\right) \\
&\leq 4ne^{-Hp^3/24} \leq 4e^{\ln n - Hp^3/24} .
\end{aligned}$$

The first equality follows from substitution of δ and T. The second inequality follows from the constraints $0 < p < 1$.

Since $\frac{H}{\ln n} = \omega(1)$ and $\left(\frac{\ln n}{H}\right)^{1/3} \ll p$ we get $Hp^3/24 \gg \ln n$. It follows that $\Pr\left[\ell_{\mathcal{R}_p}(\sigma) < T\right] \leq 4e^{\ln n - Hp^3/24} = o(1)$. □

COROLLARY 7.2.5. *The competitive ratio of the algorithm is at least* $1 - o(1)$.

CHAPTER 8

Engineering Considerations and Simulation Results

Our theoretical results show that our randomized algorithm is asymptotically optimal, but these results still leave two practical questions open. One question is how to set p in practice. The second question is how much can we gain, on "easy" sequences, over this algorithm by using algorithms that are perhaps sub-optimal in the competitive worst-case sense. We performed simulations to address the first question, which is highly relevant to implementations of the randomized algorithm. We have not yet performed simulations to compare this algorithm to heuristics that are perhaps sub-optimal.

How do we set p? We performed numerous simulations of our algorithm in order to provide an answer to this question. The simulations executed the algorithm ($\mathcal{R}_p$) on a fixed sequence $\sigma = 1, 1, 1, \ldots$ with various values of n and H, 50 times for each (n, H) pair. For $\mathcal{R}_p$, this constant sequence is as bad as any.

The results of some of these simulations are shown in Figures 8.0.1, 8.0.2, and 8.0.3. Each graph shows the results of all the simulations for a given n and a given H, for many different switching probabilities p. We have shown in Theorem 7.2.4 that the probability that the flash will not endure $T = nH/(1+p)^3$ requests is at most $4e^{\ln n - Hp^3/24}$. For p much higher than $(\ln n/H)^{1/3}$, this probability is small. The vertical line in each graph is at $p = (\ln n/H)^{1/3}$, and the grey line that extends from it to the right shows the curve $nH/(1+p)^3$. As we move away (to the right) from the vertical line, the probability that a run will not endure less than the yellow curve drops to zero. The y axis in all the graphs spans the range $[0, nH]$, so the relative height of a data point shows exactly how close to the idea endurance that simulation was.

Several facts emerge from the simulations. First, the average endurance rises with p and then drops again. Second, the variance of the endurance drops monotonically as we increase p. These effects are caused by the two sources of early wear in our algorithm. A low value of p allows some units to become much more worn out than others: the flash becomes worn out when one unit is erased $H + 1$ times, even though many of the other units are not nearly as worn out. At higher values of p, the wear evens out among the units, but the system performs on average more erasures per request.

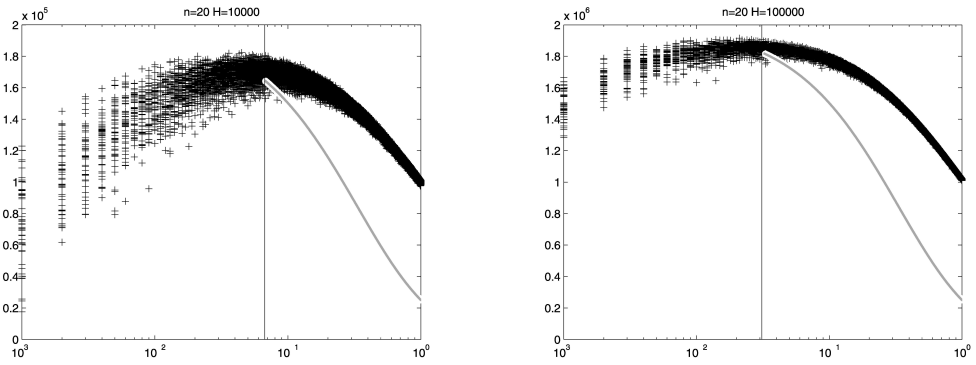

FIGURE 8.0.1. Simulation results for $n = 20$ erase units and endurance limits of $H = 10,000$ (left) and $H = 100,000$ (right). Each cross represents one simulation. The x axis shows the switching probability p that was used in the simulation, the y axis shows the number of requests that were served before one of the units was erased $H+1$ times. The y axis always extends up to exactly nH erasures, the ideal endurance. The vertical lines indicates the switching probability $p = (\ln n/H)^{1/3}$ and the grey line that extends from it down and to the right shows the curve $nH/(1+p)^3$.

For example, at $p = 1$ the best endurance that our algorithm achieves is $nH/2$; at $p = 1$ the variance is lowest, and the algorithm indeed usually achieve endurance close to $nH/2$. An intermediate value of p that balances these endurance determinants is usually best.

Third, the value $p = (\ln n/H)^{1/3}$ is clearly a good choice for p. At higher and lower values we do not see cases of significantly higher endurance but we do see cases of significantly lower endurance. System designers who do not wish to use the value $p = (\ln n/H)^{1/3}$ can use these simulations to find empirical nearly-optimal values of p.

Quantitatively, at high H/n ratios the endurance can approach 90% of the ideal endurance. As n grows the endurance drops (slowly); As H grows the endurance quickly improves. At realistic values of H, above 10,000, our algorithm usually achieves 75–90% of the ideal endurance with a good value for p.

We did not run simulations for high n and high H simultaneously, because these simulations take very long to complete.

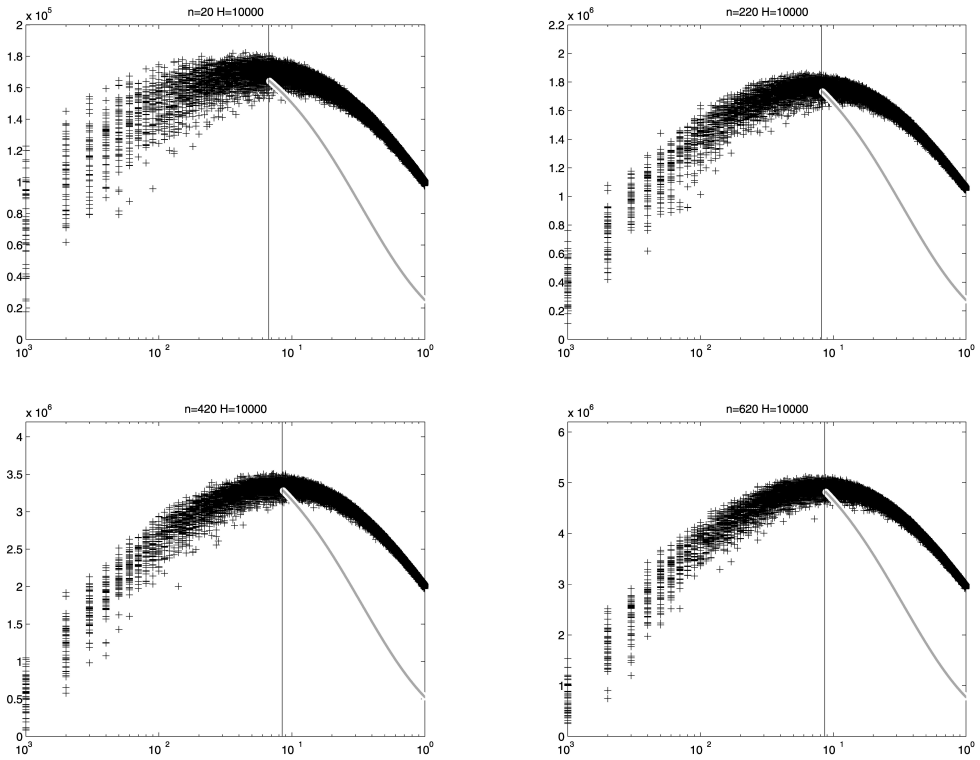

FIGURE 8.0.2. Simulation results for $H = 10,000$ and $n = 20$ (top left), $n = 220$ (top right), $n = 420$ (bottom left), and $n = 620$ (bottom right).

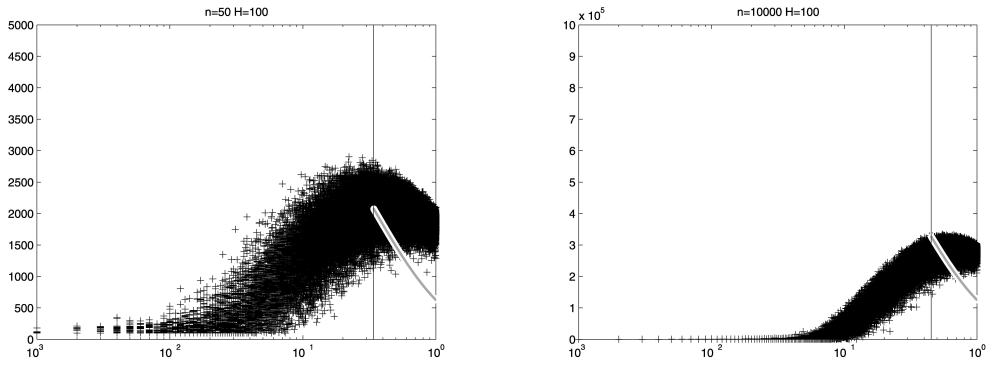

FIGURE 8.0.3. Simulation results for $H = 100$ and for $n = 50$ (left) and $n = 10,000$ (right). *These values are not realistic for flash, whose endurance limit is always 10,000 or better*, but these results show how the algorithm behaves at low H and at high $\ln n/H$ ratios.

CHAPTER 9

Fractional Wear Leveling

Until now, we have analyzed the unit case $k = 1$, in which the system writes to flash an entire erase unit in each write. Some flash devices also support fractional writes, in which erase units are k times larger than write blocks.

In this chapter, we consider the *fractional wear-leveling problem*. In this case there are m balls and n bins that can each store up to k balls. The bins store balls in fixed *slots*. A slot has three possible states: occupied by a ball, clean (in the erased state) and dirty. Algorithms are allowed to put balls only in clean bins. To change a slot state from dirty to clean, the entire bin needs to be erased.

Achieving high endurance in the fractional case is much more difficult than achieving high endurance in the unit case; the two problems are very different. Consider, for example, a request sequence with random requests. In the unit case, even a naive non-atomic write-in-place deterministic algorithm will achieve high endurance on such a sequence. The whole point of our randomized online algorithm was to introduce similar randomness into the process of serving an arbitrary sequence.

In the fractional case, the best-case scenario with a single spare bin, $m = (n - 1)k$, is $\ell = nHk$: all the balls in a given bin are requested contiguously and are moved to the spare bin, then the bin with the k dirty slots is erased, and so on. If the balls are requested such that the bins are emptied cyclically, we achieve $\ell = nHk$. On the other hand, an algorithm that always moves all the balls in a bin together (and erases an entire bin as soon as one of its slots becomes dirty) can achieve at most $\ell = nH$. Such simple algorithms are essentially unit-case algorithms and the bounds that we presented earlier apply to them. True fractional algorithms are those that try to achieve endurance close to $\ell = nHk$. Clearly, to achieve such endurance, algorithms must avoid greedy movement of balls and operate with bins that contain some dirty slots.

For such algorithms, a random request sequence is difficult to serve. A random request sequence causes slots in many bins to become dirty. When there are no more empty slots, the algorithm must erase some bin. But with high probability, no bin contains close to k dirty slots. Therefore, the

algorithm will have to erase a bin with a relatively small number of dirty slots, leading to low endurance.

CHAPTER 10

An Offline algorithm for the Fractional Problem

Clearly, $\ell_{\mathsf{opt}} \leq nHk$: there are nk slots, and each slot can endure H erasures. On the other hand, a simple lower bound is given by the same algorithm that we used in the unit bins problem. This can be done by treating all balls that are initially in the slots of the same bin as one big ball that moves between bins. As seen previously, this algorithm will achieve $\ell_{\mathsf{alg}} \geq (H-1)n$.

Obviously, the fractional unit wear leveling problem is only interesting when there are some empty bins, since when all the bins are full, the problem is equivalent to the unit wear leveling problem. First, we describe an non-atomic algorithm which for $H \gg 1$ achieves

$$\ell_{\mathsf{off}}(\sigma) \geq (1-o(1))\left(\frac{kHn}{9}\right)^{2/3} = (1-o(1))\frac{\left(\frac{1}{9}k\right)^{2/3}}{(Hn)^{1/3}}Hn$$

using a single empty bin. This algorithm is better than the naive algorithm when k is sufficiently large (specifically, when $k \geq 9\sqrt{Hn}$). We later describe two atomic variants, one with the same ℓ using $k+2$ empty bins, and another which loses a factor of $\log k$ but manages to achieve this using only three empty bins.

The idea is to split the concerns of the algorithm. We first devise an algorithm $\mathcal{N}$ that attempts to minimize the total wear (the total number of erasures). We then apply the technique of the algorithm from Section 5.1 to even the wear among the bins.

The algorithm $\mathcal{N}$ serves σ_t as follows. If there is a clean slot, it puts σ_t in it. Otherwise, it erases all the bins that contain dirty slots, and sorts all the balls stored in them in the order of their future arrival time. That is, after these erasures, there is a bin which is completely clean, and all the other erased bins are completely occupied and sorted.

For simplicity we assume that initially the bin n is the empty bin, and that whenever erasures are preformed, bin n is always erased and arranged such that after the arrangement it is the one empty bin (regardless of whether there are any dirty slots in it). This assures us that bin n is always the empty bin.

Since there are exactly k clean slots, $\mathcal{N}$ preforms erasures after each k consecutive requests. Thus, we split the executions of $\mathcal{N}$ into phases. Each

phase consists of serving k requests and performing subsequent erasures. Phase ϕ ends just before serving $\sigma_{\phi k+1}$. Let A_ϕ denote the set of erased bins at the end of the ϕth phase. Our main objective now is to bound $\sum A_\phi$. To do this, we define the following labeling scheme. A ball x is associated with a set of labels denoted by S_x. Initially all these sets are empty. They are updated between phases. After the ϕth phase, the only balls whose sets are updated are the balls in the bins of A_ϕ. First, the sets $S_{\sigma_{(\phi-1)k+1}}, \ldots, S_{\sigma_{\phi k}}$ of the requested balls becomes empty. Then, for each of ball x within the bins of A_ϕ the label (ϕ, z_x) is added to S_x, where $z_x \in \{1, \ldots, k\}$ indicates the bin order of the bin that now contains x (i.e., if x's arrival time is the jth shortest one among the balls in the bins of A_ϕ, then $z_x = \lceil j/k \rceil$). Observe that a bin that contains balls with label (ϕ, z) is not erased until all the balls with label $(\phi, z-1)$ are requested. The sets $\{S_x\}$ change during the execution; we denote the set S_x before the ϕth phase by $S_x(\phi)$.

DEFINITION 10.0.6. A ball x is *accessible* just before the ϕth phase begins if, for each label pair $(a, z) \in S_x(\phi)$, there is no other set $S_y(\phi)$ such that $(a, z') \in S_y(\phi)$ for some $z' < z - 1$.

LEMMA 10.0.7. *If all the balls in bin $i \neq n$ are inaccessible just before phase ϕ begins, then $i \notin A_\phi$.*

PROOF. Consider a ball x in bin i just before phase ϕ begins. Because x is inaccessible, there was some label $(a, z) \in S_x(\phi)$ and some other ball y such that $(a, z') \in S_y(\phi)$ for some $z' < z-1$. In particular all the k balls with label $(a, z-1)$ have not yet been requested (as requesting a ball is the only way to clear its labels). Thus, none of the k requests of the ϕth phase is for x. This is true for all the balls in bin i, so $i \notin A_\phi$. □

LEMMA 10.0.8. *If a ball x in bin $i \neq n$ is inaccessible just before phase ϕ begins due to a label (a, z), then all the balls in i have label (a, z) just before phase ϕ begins.*

PROOF. If x is inaccessible due to a label (a, z), then all the balls with label (a, z) are inaccessible. These balls were labeled with (a, z) because they were all put in a particular bin when phase a ended. Because they are all inaccessible, they must all still be in that same bin. □

We can strengthen Lemma 10.0.7.

LEMMA 10.0.9. *If there exists a ball x in bin $i \neq n$ that is inaccessible before phase ϕ, then $i \notin A_\phi$.*

PROOF. A direct corollary of Lemmas 10.0.7 and 10.0.8. □

DEFINITION 10.0.10. A bin $i \neq n$ is *erasable* just before the ϕth phase if all the balls in it at that time are accessible.

We denote by B_ϕ the set of erasable bins before ϕth phase and define $\zeta_\phi = |B_{\phi+1} \setminus B_\phi|$.

LEMMA 10.0.11. $A_\phi \setminus \{n\} \subseteq B_\phi$.

PROOF. Follows directly from the previous lemma: if a bin is erased at the end of phase ϕ, then it could not have contained an inaccessible ball at the beginning of the phase, so the bin is erasable. □

LEMMA 10.0.12. $|B_{\phi+1}| \leq |B_\phi| - |A_\phi| + 3 + \zeta_\phi$ *for any* $\phi \geq 1$.

PROOF. A bin that is erasable in the ϕth phase but was not erased (it is not in A_ϕ) is surely erasable in the $(\phi+1)$th phase. We now examine the set A_ϕ. By the previous lemma, every bin $i \neq n$ in A_ϕ was erasable before just before phase ϕ. At the end of the ϕth phase, the algorithm sorts the balls in A_ϕ according to their next arrival times. Therefore, most of the bins in this set are not erasable in the $(\phi+1)$th phase: only the two bins with the shortest arrival times are. This adds 2 to the right-hand side of the inequality. Bin n is always erased but it is not erasable (by definition): this adds 1 to the right-hand side. To bound $|B_{\phi+1}|$ we only need to add ζ_p, the number of bins that became erasable during the ϕth phase. □

The next step in the analysis is to prove a bound on $\sum_\phi \zeta_\phi$, but we first need a preliminary result.

LEMMA 10.0.13. *Let $D_1, \ldots, D_\Phi$ be Φ sets of pairs of integers (a, z), where the first component in each pair is an integer between 1 and Φ. Suppose that for $i \neq j$, there is at most one integer a that appears as the first component in a pair in D_i and in a pair in D_j. Then*

$$\left|\bigcup_{i=1}^{\Phi} D_i\right| \leq 3\Phi\sqrt{\Phi}\,.$$

PROOF. We first remove duplicate pairs from the sets: We define $D'_i = D_i \setminus (\bigcup_{j<i} D_j)$. Now each pair (a, z) that was in more than one set D_i, appears only in one of the D'_i's. Clearly, $\left|\bigcup_{i=1}^{\Phi} D_i\right| = \left|\bigcup_{i=1}^{\Phi} D'_i\right| = \sum_{i=1}^{\Phi} |D'_i|$.

Let X denote the indices of the $\sqrt{\Phi} - 1$ largest sets from $\{D'_i\}$. If for some $i \in X$ the set D'_i has at most $(3/2)\sqrt{\Phi}$ items, then clearly every set D'_j for $j \notin X$ has at most $\sqrt{\Phi}$ items. Therefore

$$\sum_{i=1}^{\Phi} |D'_i| = \sum_{i \in X} |D'_i| + \sum_{i \notin X} |D'_i| \leq |X| \cdot \Phi + (\Phi - |X|) \cdot (3/2)\sqrt{\Phi} \leq 3\Phi\sqrt{\Phi}\,.$$

Otherwise, for each $i \in X$ the set D'_i has at least $(3/2)\sqrt{\Phi}$ items. Therefore,

$$\sum_{i \in X} |D'_i| \geq (\sqrt{\Phi} - 1) \cdot (3/2)\sqrt{\Phi} = (3/2)\Phi - (3/2)\sqrt{\Phi}\,.$$

We claim that for every $R \subset \{1, \ldots, \Phi\}$ it holds

$$\sum_{i \in R} |D'_i| \leq \Phi + \binom{|R|}{2} .$$

For any $(a, z) \in D'_i$ we know that $a \in \{1, \ldots, \Phi\}$. Each such a can be represented in $\bigcup_{i \in R} D'_i$. The total number of a's which are represented by more than set is at most $\binom{|R|}{2}$ since for every pair sets D'_i and D'_j there exists at most one a such that $(a, z_1) \in D'_i$ and $(a, z_2) \in D'_j$ (for $z_1 \neq z_2$).

Let $y \notin X$. We apply the claim on $X \cup \{y\}$ to conclude

$$\sum_{i \in X \cup \{y\}} |D'_i| \leq \Phi + \binom{\sqrt{\Phi}}{2} \leq (3/2)\sqrt{\Phi} .$$

On the other hand $\sum_{i \in X} |D'_i| \geq (3/2)\Phi - (3/2)\sqrt{\Phi}$. Therefore, $\left|D'_y\right| \leq (3/2)\sqrt{\Phi}$. This holds for every $y \notin X$ Thus

$$\sum_{i=1}^{\Phi} |D'_i| = \sum_{i \in X} |D'_i| + \sum_{i \notin X} |D'_i| \leq |X| \cdot \Phi + (\Phi - |X|) \cdot (3/2)\sqrt{\Phi} \leq 3\Phi\sqrt{\Phi} .$$

□

We can now prove a bound on $\sum_\phi \zeta_\phi$.

LEMMA 10.0.14. *For any $\Phi \geq 1$ we have $\sum_{i=1}^{\Phi} \zeta_i \leq 9\Phi\sqrt{\Phi}$.*

PROOF. We wish to bound the number of events in which a bin changes its state from non-erasable to erasable. A bin becomes non-erasable when it is erased and used to store balls with label (a, z) for some $z > 2$. (The bins that are used to store balls with label (a, z) for $z = 1, 2$ are immediately erasable.) For such a bin to become erasable again, all the k balls with labels $(a, z-2)$ must be requested. Until the label $(a, z-2)$ disappears, the bin that stores balls labeled (a, z) does not become erasable again. Therefore, what we need to count to bound $\sum \zeta_i$ is the number of labels that completely disappears.

Only $k\Phi$ balls are requested during the first Φ phases. However, this does not give a bound of Φ on the number of bins that become erasable again during these Φ phases, because requested balls with more than one label may contribute to the erasability of multiple bins.

Let $C_t = S_{\sigma_t}(\lceil t/k \rceil)$ be the set of labels that ball σ_t carries at time t. The number of labels (a, z) that appear exactly k times in the C_t's is exactly the number of bins that become erasable again. Other labels, the ones that appear fewer than k times, are irrelevant and we completely ignore them in the rest of the analysis. Thus, from now on, we assume that each label appears in exactly k of the C_t's.

We claim that the total number of labels in these (reduced) C_t's is at most $9\Phi\sqrt{\Phi}$. We prove this claim by showing that there is a subset of only Φ out of the $k\Phi$ sets that contain a constant fraction of the labels, and that in that subset there are only $3\Phi\sqrt{\Phi}$ labels. Let $D = \{D_1, \ldots, D_\Phi\}$ be a random sample of the C_t's, drawn uniformly and independently (with repetitions). The probability that a particular label appears in one of the D_i's is exactly $1/\Phi$, because exactly k of the $k\Phi$ sets C_t contain that label. The probability that a particular label does not appear in any of the D_i's is, therefore,

$$\left(1 - \frac{1}{\Phi}\right)^{\Phi} \leq \frac{1}{e} < \frac{2}{3} .$$

Hence, the probability that the label does appears in some of the D_i's is bounded from below by a constant. Therefore, the expected number of labels that appear in $\cup_i D_i$ is bounded from below by a $1/3$ times the number of labels in $\cup_t C_t$. This implies that there is some specific sample $D = \{D_1, \ldots, D_\Phi\}$ in which the number of labels is at least $1/3$ fraction of the labels in the (reduced) C_t's.

We now show that there are constraints on the labels that different C_t's can contain. These constraints carry over to the sample D, and they will allow us to prove a bound on the number of labels in D and hence on $\sum \zeta_\phi$.

We say that two sets C_{t_1} and C_{t_2} are *linked by* a if $(a, z) \in C_{t_1}$ and $(a, z') \in C_{t_2}$ for some $z' \neq z$. We claim that if C_{t_1} and C_{t_2} are linked by a, then they cannot be linked by any other phase-label $b \neq a$. Suppose for contradiction that the claim is false and that the two sets are also linked by b. Without loss of generality, let $a < b$ and that $z' > z$. If time t_1 occurs after phase b ends, then $(b, ?) \notin C_{t_2}$, because until after time t_1, the ball associated with C_{t_2} is in a bin in which all the balls are labeled by (a, z'). None of these balls can be requested until after time t_1, so the ball associated with C_{t_2} cannot be labeled with b. On the other hand, if time t_1 occurs before phase b ends, then $(b, ?) \notin C_{t_1}$.

This argument essentially concludes the proof: The C_t's satisfy the mutual exclusion assumption of Lemma 10.0.13. This implies that so do the D_i's in the specific set D. Lemma 10.0.13 guarantees that the union of the D_i's does not contain too many labels. Specifically $\left|\bigcup_{i=1}^{\Phi} D_i\right| \leq 3\Phi\sqrt{\Phi}$. Since $\left|\bigcup_{i=1}^{\Phi} D_i\right| \geq \frac{1}{3}\left|\bigcup_{i=1}^{k\Phi} C_i\right|$ we conclude that $\left|\bigcup_{i=1}^{k\Phi} C_i\right| \leq 9\Phi\sqrt{\Phi}$. The lemma follows from the fact that $\sum_{i=1}^{\Phi} \zeta_i = \left|\bigcup_{i=1}^{k\Phi} C_i\right|$. □

If the number of bins that become erasable is small, the flash endures.

THEOREM 10.0.15. *If $H \gg 1$ then for $t \leq (1 - o(1))(kHn/9)^{2/3}$ the total number of erasures under this offline algorithm is $\sum_{i=1}^{n} h_i^{\mathcal{N}}(t) \leq Hn - n$.*

PROOF. It follows from Lemma 10.0.12 (by simple induction, using $|B_1| \leq n$) and from the fact that

$$|B_\Phi| \leq n - \sum_{i=1}^{\Phi-1} |A_i| + 3(\Phi - 1) + \sum_{i=1}^{\Phi} \zeta_i \ .$$

From Lemma 10.0.11 we know that $|B_\Phi| + 1 \geq |A_\Phi|$. Therefore,

$$|A_\Phi| \leq 1 + n - \sum_{i=1}^{\Phi-1} |A_i| + 3(\Phi - 1) + \sum_{i=1}^{\Phi} \zeta_i \ .$$

This implies

$$\sum_{i=1}^{\Phi} |A_i| \leq 1 + n + 3(\Phi - 1) + \sum_{i=1}^{\Phi} \zeta_i \leq 1 + n + 3(\Phi - 1) + 9\Phi\sqrt{\Phi} \ ,$$

where the last inequality follows from the previous lemma. Since $\sum_{i=1}^{\Phi} |A_i| = \sum_{i=1}^{n} h_i^{\mathcal{N}}(\Phi k)$ we get $\sum_{i=1}^{n} h_i^{\mathcal{N}}(\Phi k) \leq 1 + n + 3(\Phi - 1) + 9\Phi\sqrt{\Phi}$ and conclude that $\sum_{i=1}^{n} h_i^{\mathcal{N}}(t) \leq n + 3(t/k - 1) + 9(t/k)^{3/2}$. Thus, for $t \leq (1 - o(1))(kHn/9)^{2/3}$ it holds that $\sum_{i=1}^{n} h_i^{\mathcal{N}}(t) \leq Hn - n$. □

We now use the algorithm for unit wear-leveling problem to create an algorithm $\mathcal{N}_2$ which evens the wear among the bins. This is done using a similar reduction to the one from Section 5.2. More specifically, $\mathcal{N}_2$ first simulates $\mathcal{N}$ and generates a new *unit-case* request sequence η. To avoid confusion, we call the balls in η pseudo-balls. We construct η as follows: whenever $\mathcal{N}$ erases bin i, we add a request for pseudo ball i to η (since $\mathcal{N}$ may erase many bins at once, we impose an arbitrary order on these erasures). Now $\mathcal{N}_2$ runs the offline algorithm for the unit case on η. When the unit-case offline algorithm switches balls among two bins, $\mathcal{N}_2$ switches the corresponding actual bins. It is not hard to verify, using similar arguments to those in Section 5.2 that the wear on each bin i at time t in the execution of the unit-case offline algorithm on η is exactly the same as the wear on bin i before erasure t in the execution of $\mathcal{N}_2$ on σ.

THEOREM 10.0.16. *If $H \gg 1$ then the wear $h_i^{\mathcal{N}_2}(t)$ of any bin i at time $t = (1 - o(1))(2kHn/3)^{2/3}$ is at most H.*

PROOF. Since by the previous theorem we know that $\sum_{i=1}^{n} h_i^{\mathcal{N}}(t) \leq Hn - n$, we conclude that η contains at most $Hn - n$ requests. The theorem now follows from Theorem 5.1.6. □

The algorithm $\mathcal{N}$ is clearly not atomic, since whenever erasures are preformed, $\mathcal{N}$ first reads the entire contents of several bins into a separate staging area, erases these bins, and rearranges the data into flash bins. However, it is not to hard to convert $\mathcal{N}$ into an atomic algorithm using $k + 2$ empty bins.

Let $\mathcal{A}$ denote the following algorithm. As in the non atomic case, $\mathcal{A}$'s execution is divided into phases. Each step consists of serving k requests and performing subsequent erasures. First, let us assume there are only $k+1$ empty bins (the $(k+2)$th empty bin will be used for wear leveling). The definition of $\mathcal{A}$ will assure that at the beginning of each phases there will be exactly $k+1$ empty bins. At the beginning of each phases, $\mathcal{A}$ chooses an arbitrary empty bin. The following k requests will be served using the k clean slots in that bin. After these requests have been served, the set D of bins that contains a dirty slot, is of size at most k . Thus, the total number of balls in D is at most $k(k-1)$ (since each of them contains at least 1 dirty slot). Therefore the remaining k empty bins will suffice to hold all the balls of D and the balls of the first used empty bin. $\mathcal{A}$ continues by sorting the balls of D and the first empty bin in the empty bins in the order of their future arrival time. Finally, $\mathcal{A}$ erases all the bins of D, and the first empty bin (which are now completely dirty and empty).

Now, as in the non-atomic case, we can create an algorithm $\mathcal{A}_2$ such that $\mathcal{A}_2$ simulates $\mathcal{A}$ and uses the atomic unit wear leveling algorithm to even the wear among the bins. This is done using the unused $(k+2)$th empty bin to function as the empty bin in the atomic unit wear leveling algorithm.

THEOREM 10.0.17. *If $H \gg 1$ then the wear $h_i^{\mathcal{A}_2}(t)$ of any bin i at time $t = (1-o(1))(2kHn/3)^{2/3}$ is at most H.*

PROOF. This follows from the same arguments as in the non atomic case and from Theorem 5.2.7. □

If k is large (this is when $\mathcal{N}$ is effective), $\mathcal{A}_2$ needs many empty bins. We show that it is possible to trade off these extra bins for somewhat reduced endurance. At the end of a phase, $\mathcal{N}$ needs to sort the balls in the dirty bins. We perform the sorting using the merge-sort algorithm, using two extra bins to hold the partial runs of sorted balls that the algorithm constructs. It is not hard to see that two extra bins are always sufficient, and that the sorting algorithm performs $O(k \log k)$ erasures. Let $\mathcal{A}'$ denote the described algorithm. Again we can use the atomic unit wear leveling to even the wear among the bins, with the help of an extra empty bin. We obtain an atomic algorithm $\mathcal{A}'_2$ such that:

THEOREM 10.0.18. *If $H \gg 1$ then the wear $h_i^{\mathcal{A}'_2}(t)$ of any bin i at time $t = \Omega\left((2kHn/3)^{2/3}/\log k\right)$ is at most H.*

The same idea can be used to trade off any number of extra bins for better endurance using multiway merge-sort.

CHAPTER 11

The Deterministic Online Fractional Problem

The deterministic online case for the fractional wear leveling problem is similar to the unit case. The algorithm we analyze is the following. Given a request for ball x the algorithm puts x in an arbitrary clean slot. If there are no clean slots, the algorithm erases all the bins (returning each ball to the slot it was taken out of). This algorithm may seem too naive and inefficient, but our analysis shows that it is optimal (under a deterministic online analysis).

THEOREM 11.0.19. *The wear of any bin after* $(nk-m+1)(H+1)$ *requests are served is at most* H.

PROOF. The wear of all the bins increases by 1 every $nk-m+1$ requests. □

This algorithm is non-atomic. However, it can easily be converted to atomic using a single empty bin. The empty bin is used to preform the erasure of all the bins, one by one. Therefore, the atomic algorithm can serve $((n-1)k-m+1)(H+1)$ requests without exceeding the endurance of the flash.

THEOREM 11.0.20. *For every deterministic algorithm* α *(even if* α *is non-atomic) there exists a sequence* σ *such that* $\ell_\alpha(\sigma) \leq (nk-m+1)(H+1)$.

PROOF. Given α, we will generate σ as follows. The adversary fixes an arbitrary set T of $nk-m+1$ slots. At every time during the execution of α, at least one slot in T must contain a ball, since there are only $nk-m$ slots that do not contain a ball. The sequence σ always requests a ball that is in one of the slots of T. This ensures that after $(nk-m+1)(H+1)+1$ requests at least one of the slots of T is emptied more then H times, and therefore its bin is erased more the H times. Thus, $\ell_\alpha(\sigma) \leq (nk-m+1)(H+1)$. □

Clearly, our naive deterministic algorithm is optimal in the worst-case sense.

CHAPTER 12

The Randomized Online Fractional Problem

In this chapter we present a simple randomized online algorithm for the fractional problem, and a lower bound for such algorithms. The main ingredient that we analyze is, again, the number of erasures preformed by the algorithm.

The algorithm we analyze is, as in the offline case, a composition of an erasure-minimization algorithm with a wear-leveling one. The erasure minimization algorithm serves request σ_t as follows. If there is a clean slot, σ_t is put in it. Otherwise the algorithm erases the bin with the largest number of dirty slots and then puts σ_t in it. This is a deterministic algorithm that might suffer from poor wear leveling, so we combine it with the randomized *unit wear leveling* algorithm to even the wear among the bins. (That is, whenever a bin is erased, with some probability p, its entire contents is switched with some other random bin). Clearly the total number of slots cleaned by each erasure is at least $\lceil (nk-m+1)/n \rceil$. Thus we obtain an algorithm $\mathcal{R}$ such that:

- For $\Omega(\frac{1}{\ln \ln n}) \leq \frac{H}{\ln n} = O(1)$ we get, for any sequence σ, that
$$\Pr\left[\ell_{\mathcal{R}}(\sigma) < \frac{n^{1-\frac{c}{H}} \ln n}{\lceil (nk-m+1)/n \rceil}\right] = o(1) \ .$$
- For $\frac{H}{\ln n} = \omega(1)$ we get, for any sequence σ, that
$$\Pr\left[\ell_{\mathcal{R}_p}(\sigma) < \frac{nH(1-o(1))}{\lceil (nk-m+1)/n \rceil}\right] = o(1) \ .$$

We now turn to proving a lower bound for such algorithms. The bound we prove depends on the number of empty slots s. In particular we demand that $s < nk/2$, since if half of the slots are empty even a deterministic algorithm can achieve high performance.

THEOREM 12.0.21. *For every randomized online algorithm α (even if α is non-atomic) for the fractional wear leveling problem with $s = s(n)$ empty slots, such that $s \ll Hn$ and $s < nk/2$, there exists a constant $c > 0$ and a sequence σ such that $\boldsymbol{E}[\ell_\alpha(\sigma)] < cHn \cdot \boldsymbol{E}[ri(n/2, 2s)]$ (ri refers to the random-increment game). In particular:*

- *If $s = n^{1-\epsilon}$ for some constant $0 < \epsilon < 1$ then there exists a constant $c > 0$ and σ such that $\boldsymbol{E}[\ell_\alpha(\sigma)] < cHn$.*

- *If* $\frac{n}{polylog(n)} \leq s \ll n \log n$ *then there exists a constant* $c > 0$ *and* σ *such that*

$$\boldsymbol{E}\left[\ell_\alpha(\sigma)\right] < cHn \cdot \frac{\log n}{\log\left(\frac{n \log n}{s}\right)} .$$

- *If* $s = \Omega(n \log n)$ *then there exists a constant* $c > 0$ *and* σ *such that* $\boldsymbol{E}\left[\ell_\alpha(\sigma)\right] < cHs$.

PROOF. The proof analyzes the number of erasures that α performs on a sequence σ whose elements are chosen uniformly and independently at random. We split the execution into phases, each consists of serving consecutive $2s$ requests. The $2s$ requests of phase i cause $2s$ slots to become dirty (perhaps with repetitions). At the end of the phase, as in any point, there are at most s dirty slots. Therefore, at least s of the slots that became dirty due to σ during the phase became clean again before it ended.

Since $s < nk/2$, there are at least $nk/2$ balls. On the other hand, each bin contains at any time, at most k balls. Therefore, the probability that a single request will make a slot dirty in some fixed bin i is at most $2/n$. Hence, the expectation of the maximum number of slots that become dirty due to σ in a single bin is at most $\mathrm{E}\left[\mathrm{ri}(n/2, 2s)\right]$ (The distribution of the maximum number of dirty slots due to σ is slightly lower than $\mathrm{ri}(n/2, 2s)$ since whenever a slot becomes dirty, the number of balls in its bin decreases, so future requests are less likely to request a ball in this bin. However, $\mathrm{E}\left[\mathrm{ri}(n/2, 2s)\right]$ is surely an upper bound for the expectation of such a distribution.)

Thus, with probability at least $1/2$ there will not be a bin which has more than $2 \cdot \mathrm{E}\left[\mathrm{ri}(n/2, 2s)\right]$ dirty slots. Hence, the probability that in phase i the algorithm erases only bins with less than $2 \cdot \mathrm{E}\left[\mathrm{ri}(n/2, 2s)\right]$ dirty slots (due to σ) is at least $1/2$. The algorithm must clean at least s dirty slots (due to σ) during the phase, so the probability that it performs at least $s/\left(2 \cdot \mathrm{E}\left[\mathrm{ri}(n/2, 2s)\right]\right)$ erasures is at least $1/2$.

We define indicators $\{X_i\}$ such that $X_i = 1$ if and only if there are at least $s/\left(2 \cdot \mathrm{E}\left[\mathrm{ri}(n/2, 2s)\right]\right)$ erasures during ith phase. We have just shown that $\Pr[X_i = 0] \leq 1/2$, where the probability is taken over both $\alpha's$ coins and the sequence σ.

Let e_ϕ denote the total number of erasures until the ϕth phase. It follows that $e_\phi \geq \sum_{i=1}^{\phi} X_i \cdot s/\left(2 \cdot \mathrm{E}\left[\mathrm{ri}(n/2, 2s)\right]\right)$. By Chernoff bound we get $\Pr\left[e_\phi \leq \frac{1}{4} \cdot \phi s/\left(2 \cdot \mathrm{E}\left[\mathrm{ri}(n/2, 2s)\right]\right)\right] \leq 2^{-\Omega(\phi)}$. (We can apply the Chernoff bound even though the X_i's are not independent because clearly we can replace them with independent indicator variables with mean $1/2$).

Let $y = 16Hn \cdot \mathrm{E}\left[\mathrm{ri}(n/2, 2s)\right]$. We know that

$$\Pr\left[\ell_\alpha(\sigma) > y\right] \leq \Pr\left[e_{y/2s} \leq Hn\right] ,$$

since for $\ell_\alpha(\sigma)$ to be greater than y, the total number of erasures until phase $y/2s$ cannot exceed the total number of erasures the bins can endure. Since

$$\frac{1}{4} \cdot y \cdot (s/\left(2 \cdot \mathbf{E}\left[\mathbf{ri}(n/2, 2s)\right]\right)) = \frac{16Hn}{16} = Hn \ ,$$

we get

$$\Pr\left[e_y \leq Hn\right] = \Pr\left[e_{y/2s} \leq \frac{1}{4} y \cdot (s/\left(2 \cdot \mathbf{E}\left[\mathbf{ri}(n/2, 2s)\right]\right))\right] = 2^{-\Omega(y/2s)} = o(1).$$

Were the last equality follows from the fact that $s \ll Hn$. Thus $\mathbf{E}\left[\ell_\alpha(\sigma)\right] < 16Hn \cdot \mathbf{E}\left[\mathbf{ri}(n/2, 2s)\right]$, where the expectation is taken over both $\alpha's$ coins and the sequence σ. Therefore there exists a sequence σ' such that $\mathbf{E}\left[\ell_\alpha(\sigma')\right] < 16Hn \cdot \mathbf{E}\left[\mathbf{ri}(n/2, 2s)\right]$.

The further conclusions follow from the following facts:

- $\mathbf{E}\left[\mathbf{ri}(n/2, 2n^{1-\epsilon})\right] = \Theta(1)$ (Lemma 2.13 in [**19**]).
- For $\frac{n}{\text{polylog}(n)} \leq s \ll n \log n$ it holds that

  $$\mathbf{E}\left[\mathbf{ri}(n/2, 2s)\right] = \Theta\left(\frac{\log n}{\log\left(\frac{n \log n}{s}\right)}\right)$$

  (Theorem 1 in [**20**]).
- For $s = \Omega(n \log n)$ it holds that $\mathbf{E}\left[\mathbf{ri}(n/2, 2s)\right] = \Theta(s/n)$ (Theorem 1 in [**20**]).

□

CHAPTER 13

Conclusions

The wear-leveling problem in flash memories has been studied since at least 1993, mostly in industry. In this thesis we have analyzed two variants of this problem, the unit wear-leveling problem and the fractional wear-leveling problem.

We have shown that in the worst adversarial case, deterministic wear-level algorithms perform poorly. Their endurance is proportional to the number of unused flash memory cells. In particular, when there are few unused cells (the typical case; people buy memory to use it), their performance is very poor; the flash can wear out when most of its cells are not worn out at all. The use of such algorithms in practice [**1, 2, 13, 10, 6, 12, 16, 17, 21, 4, 3, 22, 9, 14, 23, 7, 8, 15**] is probably due to the fact that actual request sequences are oblivious to the wear-leveling algorithm rather than adversarial. For example, it was shown in [**11**] that one common deterministic wear-leveling algorithm performs well on realistic benchmarks.

We showed that a randomized wear-leveling algorithm due to Ban [**5**] achieves asymptotically-optimal endurance in the unit case. When the guaranteed number of erasures per memory cell is large (typical in practice), this algorithm achieves not only nearly-optimal competitive performance relative to any online wear-leveling algorithm, but asymptotically optimal performance relative to the ideal endurance. This implies not only that this algorithm is highly effective in practice, but also that there is no point in trying to guess the yet-unseen suffix of the request sequence.

The unit wear-leveling problem models two situations. It obviously models situations in which the size of write blocks and the size of erase units are the same. It also models situations in which erase units are larger than write blocks, but the system separates the allocation policy, cleaning policy, and wear-leveling policy. Suppose that we write to flash small fixed- or variable-size blocks. Many systems separate the allocation policy, which decides in which clean slot to store the new contents of a logical block, the cleaning policy, which decides which erase unit to erase to reclaim the space occupied by dirty slots, and the wear-leveling policy, which decides whether to erase additional units when the cleaning policy decides to clean a particular unit. In such systems, the cleaning policy essentially generates a

request sequence for a unit-case wear-leveling problem. TFFS is a typical example of such a system [**11**].

When the flash-management software treats the allocation, cleaning, and wear-leveling issues in a single policy, that policy must solve a fractional wear-leveling problem. This problem is more difficult to analyze. The problem comes in two flavors: that of fixed-size write blocks and that of variable-size write blocks (Actual systems exhibit both flavors). We have analyzed the fixed-size fractional problem. We have shown that online algorithms, whether deterministic or randomized, cannot achieve high endurance unless many memory cells remain unused. This problem is caused not by the difficulty of leveling the wear, but by the difficulty of reducing the number of erasures significantly below one erasure per request. On the other hand, we have shown that an offline algorithm can perform better (although our endurance bound for the offline problem is still far from the ideal nHk).

These findings concerning the fractional problem have two practical implications. First, they imply that separating the allocation and cleaning policies from the wear-leveling problems is reasonable: an online algorithm cannot gain much by addressing all of them together. Second, our nontrivial bound for the offline problem shows that heuristics that attempt to guess the future of the request sequence might be able to improve endurance when all three policies are treated as one.

Bibliography

[1] Mahmud Assar, Siamack Nemazie, and Petro Estakhri. Flash memory mass storage architecture incorporation wear leveling technique, December 1995.

[2] Mahmud Assar, Siamack Nemazie, and Petro Estakhri. Flash memory mass storage architecture incorporation wear leveling technique without using CAM cells, January 1996.

[3] Amir Ban. Flash file system, April 1995.

[4] Amir Ban. Flash file system optimized for page-mode flash technologies, August 1999.

[5] Amir Ban. Wear leveling of static areas in flash memory, May 2004.

[6] Ricardo H. Bruce, Ronaldo H. Bruce, Earl T. Cohen, and Allan J. Christie. Unified re-map and cache-index table with dual write-counters for wear-leveling of non-volitile flash ram mass storage, December 1999.

[7] M.-L. Chiang and R.-C. Chang. Cleaning policies in mobile computers using flash memory. *The Journal of Systems and Software*, 48(3):213–231, 1999.

[8] Mei-Ling Chiang, Paul C.H. Lee, and Reui-Chuan Chang. Using data clustering to improve cleaning performance for flash memory. *Software—Practice and Experience*, 29(3), 1999.

[9] Raz Dan and Jeff Williams. A TrueFFS and FLite technical overview of M-Systems' flash file systems. Technical Report 80-SR-002-00-6L Rev. 1.30, M-Systems, March 1997.

[10] Petro Estakhri, Mahmud Assar, Robert Reid, Alan, and Berhanu Iman. Method of and architecture for controlling system data with automatic wear leveling in a semiconductor non-volitile mass storage memory, November 1998.

[11] Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37:138–163, 2005.

[12] Sang-Wook Han. Flash memory wear leveling system and method, January 2000.

[13] Edwin Jou and James H. Jeppesen III. Flash memory wear leveling system providing immediate direct access to microprocessor, October 1996.

[14] Atsuo Kawaguchi, Shingo Nishioka, and Hiroshi Motoda. A flash-memory based file system. In *Proceedings of the USENIX 1995 Technical Conference*, pages 155–164, New Orleans, Louisiana, January 1995.

[15] Han-Joon Kim and Sang-Goo Lee. An effective flash memory manager for reliable flash memory space management. *IEICE Transactions on Information and Systems*, E85-D(6):950–964, 2002.

[16] Karl M. J. Lofgren, Robert D. Norman, Gregory B. Thelin, and Anil Gupta. Wear leveling techniques for flash EEPROM systems, June 2000.

[17] Karl M.J. Lofgren, Robert D. Norman, B. Thelin, Gregory, and Anil Gupta. Wear leveling techniques for flash EEPROM systems, July 2003.

[18] C. McDiarmid. *Probabilistic Methods for Algorithmic Discrete Mathematics*, chapter Concentration, pages 195–248. Springer Verlag, 1998.

[19] Michael Mitzenmacher. *The Power of Two Choices in Randomized Load Balancing*. PhD thesis, University of California, Berkeley, 1996.

[20] Martin Raab and Angelika Steger. "balls into bins" - a simple and tight analysis. In *RANDOM '98: Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 159–170, London, UK, 1998. Springer-Verlag.

[21] Steven E. Wells. Method for wear leveling in a flash EEPROM memory, August 1994.

[22] David Woodhouse. JFFS: The journaling flash file system. Presented in the Ottawa Linux Symposium, July 2001 (no proceedings); a 12-page article available online from `http://sources.redhat.com/jffs2/jffs2.pdf`, July 2001.

[23] Michael Wu and Willy Zwaenepoel. eNVy: a non-volatile, main memory storage system. In *Proceedings of the 6th international conference on Architectural support for programming languages and operating systems*, pages 86–97. ACM Press, 1994.