Competitive Analysis of Flash-Memory Algorithms

Avraham Ben-Aroya and Sivan Toledo

School of Computer Science, Tel-Aviv University
{abrhambe, stoledo}@tau.ac.il

Abstract. The cells of flash memories can only 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 interesting online problems called *wear-leveling problems*. We show that a simple randomized algorithm for one problem is essentially optimal. For a more difficult problem, we show that clever offline algorithms can improve upon naive approaches, but online algorithms essentially cannot.

1 Introduction

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, flash memory cells can be erased only a limited number of times, between 10,000 and 1,000,000, after which they wear out and become unusable.

Writing to flash involves two separate operations: erasures and programming. An erasure sets all the bits in a range of cells to '1'. These ranges are called *erase units* and are usually uniform in size. We denote the number of erase units by n. The programming operation writes a given bit sequence to an erase unit, or to a part of an erase unit by clearing some of the 1's. We assume in this paper that the computer system always programs fixed-length sequences called *blocks*. We denote the number of blocks that fit within an erase unit by k. That is, each erase unit is divided into k *slots* that can each store a single block. A slot that has been programmed cannot be programmed again until the entire erase unit is erased (there are exceptions to this rule, but they are beyond the scope of this paper). Both $k = 1$ and higher values of k occur in practice.

Clever management of a flash device can dramatically extend its functional life span. Consider a device with n erase units that can each be erased H times, which are not divided into slots (that is, $k = 1$). We consider the device useless when one of the n cells exceeds the wear limit H (our results justify this assumption). 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 blocks from the physical location of the slots that contain them. The computer system views the flash device as a store of $m \leq nk$ fixed-size blocks named 1 through m. The system uses the flash by issuing a sequence of read requests and write requests. Read requests are irrelevant to endurance so we ignore them. Write requests require the flash memory manager to store the new content of a named block and to return it in the future. Initially, each data block is stored in some slot. These slots are *occupied*. The flash memory manager serves a write request by performing a sequence of erasure and programming operations. In this sequence, blocks can only be written to *clean* slots (slots that have not been written to since the containing erase unit was erased), not to *dirty* slots (slots that contain obsolete data). When a unit is erased, all its contents are lost, and all its slots become clean. 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 slot. 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 manager begins the sequence by marking the slot that contains the old copy of the block as obsolete. If the manager wishes to rearrange additional blocks, it reads them into volatile memory (RAM) and marks the slots that contained them as obsolete. Next, the manager can erase units that contain no occupied slots, only clean and dirty ones. Finally, the manager writes all the blocks that are in volatile memory, including the updated block that initiated the sequence, into erased slots. If atomicity is required, or if the amount of RAM is limited, update sequences are more complex. The mapping issue (remembering where each block is stored) is largely orthogonal to the endurance issue, and we ignore it in this paper.

Starting around 1993, a variety of wear-leveling techniques have been proposed, mostly in patents [1, 4, 5, 6, 7, 9, 10, 11, 12, 13]; for details about these techniques and about other flash-management techniques, see [8]. In this paper, we present a competitive analysis of online wear-leveling policies, including of patented randomized 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.

When the request series begins, we have m occupied slots and $nk - m$ clean slots. Thus, the manager 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 manager 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 offline 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 $\min_\sigma \ell_\alpha(\sigma)/\ell_{\mathsf{opt}}(\sigma)$, 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 randomized then we replace $\ell_\alpha(\sigma)$ by the expected length that α can serve.

The paper is organized as follows. Section 2 analyzes the case $k=1$ and Section 3 analyzes the case $k>1$. It turns out that these two cases are quite different and are governed by different issues. Each of these sections describes an effective offline algorithm, bounds on the competitiveness of deterministic and randomized online algorithms, and online algorithms that match most of the bounds. Section 4 presents our conclusions. Due to lack of space, most of the proofs have been omitted; see [3] for the proofs, for additional simulation results, and for a fuller discussion of implications of the results.

2 Single-Slot Units

We begin the analysis with single-slot erase units ($k=1$). This case models at least two real-world situations: flash devices that limit programming operations to entire erase units, and flash devices that allow variable-size programming operations, which usually leads system designers to use single-slot wear-leveling algorithms.

In this case we can simplify the rules. First, we refer only to (erase) units, not to slots. Second, units can be in only two states, clean and occupied (and not dirty): We immediately erase a unit when a block is moved from it. Any result for this simplified model applies to the full model up to a change of ± 1 to H.

2.1 Deriving Atomic Policies from Non-atomic Ones

We present a method that allows us to separate the atomicity concern from the wear-leveling concern. This method transforms many non-atomic algorithms, including all the algorithms in this section, into atomic ones with exactly the same endurance.

We denote by $h_i(t)$ the number of erasures that the algorithm already performed on unit i immediately before serving request number t. We call $h_i(t)$ the *wear* of i at time t. We denote by σ_t the index of the block requested by the tth request in the sequence σ.

Theorem 1. *Let $\mathcal{N}$ be a non-atomic algorithm for n blocks and n units that serves a request sequence η by either putting η_t back in its unit or by switching η_t with some other block. We can derive from $\mathcal{N}$ an atomic algorithm $\mathcal{A}$ for $n-1$ blocks and n units. 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.*

The transformation simulates a fixed trivial online algorithm $\mathcal{T}$ on σ and uses its state (the block-to-unit mapping) to define η. Then $\mathcal{N}$ is simulated on η. The action of $\mathcal{N}$ on η_t, which is always one of two possible actions, is used to deterministically define the action of $\mathcal{A}$ on σ_t. Therefore, if $\mathcal{A}$ is randomized, then $\mathcal{N}$ essentially uses the coin tosses of $\mathcal{A}$, and the result $h_i^{\mathcal{A}(\sigma)}(t) = h_i^{\mathcal{N}(\eta)}(t)$ holds for every sequence of coin tosses. Thus, the theorem implies that the endurance of $\mathcal{A}$ and $\mathcal{N}$ is the same, both in the worst-case sense and in probabilistic senses.

2.2 Offline Algorithms

The best-case endurance (for "easy" sequences) is $\ell(\sigma) = nH$. Offline algorithms can achieve almost this best-case endurance.

Theorem 2. *There is an atomic offline algorithm for which the wear $h_i(t)$ of any unit i at time $t = Hn - n + 1$ is at most H, for any sequence σ, even if $m = n - 1$. For $m = n$ there is a non-atomic algorithm that achieves this endurance.*

This implies that an offline algorithm can always achieve $\ell_{\mathsf{off}}(\sigma) \geq n(H - 1)$. Since $H > 10,000$, the offline endurance is exceedingly close to the best-case endurance nH.

The non-atomic offline algorithm works as follows (the atomic one uses Theorem 1). Normally, the algorithm serves a request to block x stored in unit 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 unit j. To decide whether to switch i with j, the algorithm counts the number of remaining requests to all the blocks, but only up to request number $Hn-n$. It switches if the number of remaining requests to some block y stored in unit j matches exactly the number of erasures left for unit i, *or* if the number of remaining requests to x matches exactly the number of erasures left for unit j. The algorithm performs at most n such switches on a given sequence. Notice that a switch performs two erasures, while a write in-place performs only one.

2.3 Deterministic Online Algorithms

The endurance of deterministic algorithms depends entirely on the number $n-m$ of extra erase units. An online algorithm can achieve this endurance by always putting the requested block in the least-worn out empty unit.

Theorem 3. *Under this deterministic online algorithm, the wear of any unit after $(n - m + 1)H$ requests is at most H.*

This is as good as any deterministic algorithm can achieve.

Theorem 4. *For every deterministic algorithm α (even if α is non-atomic) there exists a sequence σ such that $\ell_\alpha(\sigma) \leq (n-m+1)H$.*

In spite of this pessimistic competitive result, many flash-based systems use deterministic algorithms. This is due to the fact that in practice the request sequence is oblivious to the mapping and such algorithms usually work well.

2.4 Randomized Online Algorithms

We analyze a randomized algorithm called $\mathcal{R}_p$, which was patented by Amir Ban along with several other wear-leveling algorithms [2]. The algorithm that we analyze corresponds to Claims 2.c, 3, and 4.II in [2]. It serves a request to block x using the following rules:

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

This algorithm is not atomic, but it satisfies the conditions of Theorem 1, so we can easily derive an atomic variant with the same competitive ratio. In the analysis of the algorithm, we assume that the blocks are initially stored in random units.

For $p = 1$, the behavior of this algorithm is fairly simple. Since the block that was requested is always switched with a random block, uniformly, an adversary has no useful information about which block is in which unit. Therefore, any request sequence is equivalent to a random request sequence. For large H, the wear of all the units under a random request sequence is roughly the same. Since serving each request usually costs the algorithm two erasures (if m is close to n), the endurance should be close to $nH/2$. A full analysis of $\mathcal{R}_1$ is not difficult.

However, a small p can improve the endurance and bring it close to nH. Figure 1 shows the results of simulations of $\mathcal{R}_p$. For given n and H, we simulated $\mathcal{R}_p$ with several values of p, 50 times for each p, with a constant request sequence. Each cross on the graph indicates the length of the sequence that a particular run was able to serve. The results clearly show that the simple variant $\mathcal{R}_1$ achieves endurance of roughly $nH/2$, but for a small p, the algorithm $\mathcal{R}_p$ can get close to nH. The main goal of our analysis, which is more complex than the analysis of $\mathcal{R}_1$, is to fully analyze this phenomenon and to provide guidelines for the choice of p.

A small p only helps, however, if H is large. If H is small, $\mathcal{R}_1$ is optimal. We begin with results that bound the performance of randomized algorithms for small H and that show that $\mathcal{R}_1$ is optimal in this regime. The results from here on are asymptotic with respect to a growing n.

Theorem 5. *For any 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 $\Pr\left[\ell_\alpha(\sigma) < n^{1-c/H}\ln n\right] \geq 1 - o(1)$. (The constants within the small-o, as well as c, depend on e).*

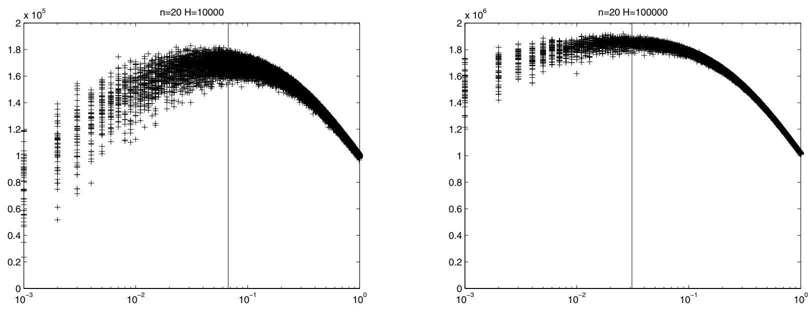

Fig. 1. Simulation results for $n = 20$ erase units and endurance limits of $H = 10,000$ (left) and $H = 100,000$ (right). Each cross represents one simulation. The x axis shows the switching probability p that was used in the simulation, the y axis shows the number of requests that were served before one of the units was erased $H + 1$ times. The y axis always extends up to exactly nH erasures, the ideal endurance. Theorem 7 shows that for switching probabilities far from 1 but significantly larger than $p = (\ln n/H)^{1/3}$ (indicated in the graphs by the vertical line), the endurance is nearly ideal.

Theorem 6. *For H in the range*

$$\Omega(\ln n/\ln\ln n) \leq H \leq O(\ln n) ,$$

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

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

When H is large, a good choice for p brings $\ell_{\mathcal{R}_p}(\sigma)$ almost to nH.

Theorem 7. *When $H = \omega(\ln n)$, for any $(\ln n/H)^{1/3} \ll p \ll 1$ and for every request sequence σ,*

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

3 Fractional Unit Wear Leveling

Until now, we have analyzed the single-slot case $k = 1$. Some flash based systems support fixed-size fractional writes, in which erase units are k times larger than write blocks.

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

Achieving high endurance in the fractional case is more difficult than achieving high endurance in the single-slot case; the two problems are very different. Consider, for example, a request sequence with random requests. In the single-slot case, even a naive non-atomic write-in-place deterministic algorithm achieves 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. But in the fractional problem, a random request sequence is difficult to serve, because it causes slots in many units to become dirty. When there are no more empty slots, the algorithm must erase some unit. But with high probability, no unit contains close to k dirty slots. Therefore, the algorithm will need to erase a unit with a relatively small number of dirty slots, leading to low endurance.

3.1 An Offline Algorithm for the Fractional Problem

Clearly, $\ell_{\mathsf{opt}} \leq (nk - m) + nHk$. On the other hand, a simple lower bound is given by the same algorithm that we used in the single-slot unit problem. This can be done by treating all blocks that are initially in the slots of the same unit as one big block that moves between units. 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 spare empty units, since when all the units are full, the problem is equivalent to the single-slot 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 unit. 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 units, and another which loses a factor of $\log k$ but uses only three empty units.

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 wear-leveling policy from Section 2.2 to even the wear among the units.

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 units that contain dirty slots, and sorts all the blocks stored in them in the order of their future arrival time. That is, after these erasures, there is a unit which is completely clean, and all the other erased units are completely occupied and sorted.

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

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 units 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 block 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 blocks whose sets are updated are the blocks in the units of A_ϕ. First, the sets $S_{\sigma_{(\phi-1)k+1}}, \ldots, S_{\sigma_{\phi k}}$ of the requested blocks becomes empty. Then, for each of block x within the units of A_ϕ the label (ϕ, z_x) is added to S_x, where $z_x \in \{1, \ldots, k\}$ indicates the unit order of the unit that now contains x (i.e., if x's arrival time is the jth shortest one among the blocks in the units of A_ϕ, then $z_x = \lceil j/k \rceil$). Observe that a unit that contains blocks with label (ϕ, z) is not erased until all the blocks 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 $S_x(\phi)$.

We say that a block 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 $z' < z - 1$. A unit $i \neq n$ is *erasable* just before the ϕth phase if all the blocks in it at that time are accessible. We denote by B_ϕ the set of erasable units before ϕth phase and define $\zeta_\phi = |B_{\phi+1} \setminus B_\phi|$.

Lemma 1. *$A_\phi \setminus \{n\} \subseteq B_\phi$.*

Lemma 2. *$|B_{\phi+1}| \leq |B_\phi| - |A_\phi| + 3 + \zeta_\phi$ for any $\phi \geq 1$.*

Proof. A unit 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 unit $i \neq n$ in A_ϕ was erasable before just before phase ϕ. At the end of the ϕth phase, the algorithm sorts the blocks in A_ϕ according to their next arrival times. Therefore, most of the units in this set are not erasable in the $(\phi+1)$th phase: only the two units with the shortest arrival times are. This adds 2 to the right-hand side of the inequality. Unit 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_ϕ, the number of units that became erasable during the ϕth phase. □

Lemma 3. *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 such that $(a, z_1) \in D_i$ and $(a, z_2) \in D_j$ for some $z_1 \neq z_2$ (i.e. a appears as the first component in a pair in D_i and in a pair in D_j). Then $\left|\bigcup_{i=1}^{\Phi} D_i\right| \leq 3\Phi\sqrt{\Phi}$.*

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

Lemma 4. *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 unit changes its state from non-erasable to erasable. A unit becomes non-erasable when it is erased and used to store blocks with label (a, z) for some $z > 2$. (The units that are used to store blocks with label (a, z) for $z = 1, 2$ are immediately erasable.) For such a unit to become erasable again, all the k blocks with labels $(a, z-2)$ must be requested. Until the label $(a, z-2)$ disappears, the unit that stores blocks 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 $k\Phi$ blocks are requested during the first Φ phases. However, this does not give a bound of Φ on the number of units that become erasable again during these Φ phases, because requested blocks with more than one label may contribute to the erasability of multiple units.

Let $C_t = S_{\sigma_t}(\lceil t/k \rceil)$ be the set of labels that block 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 units 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.

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 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 $z' > z$. If time t_1 occurs after phase b ends, then $(b, ?) \notin C_{t_2}$, because until after time t_1, the block associated with C_{t_2} is in a unit in which all the blocks are labeled by (a, z'). None of these blocks can be requested until after time t_1, so the block associated with C_{t_2} cannot be labeled with b. On the other hand, if time t_1 occurs before phase b ends, then $(b, ?) \notin C_{t_1}$.

Therefore, the C_t's satisfy the mutual exclusion assumption of Lemma 3. This implies that so do the D_i's in the specific set D. Lemma 3 guarantees that $\left|\bigcup_{i=1}^{\Phi} D_i\right| \leq 3\Phi\sqrt{\Phi}$. 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 units that become erasable is small, the flash endures.

Theorem 8. *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 2 (by simple induction, using $|B_1| \leq n$) that

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

From Lemma 1 we know that $|B_{\Phi}| + 1 \geq |A_{\Phi}|$. 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 the single-slot unit wear-leveling problem to create an algorithm $\mathcal{N}_2$ which evens the wear among the units. $\mathcal{N}_2$ first simulates $\mathcal{N}$ and generates a new *single-slot* request sequence η. To avoid confusion, we call the blocks in η pseudo-blocks. We construct η as follows: whenever $\mathcal{N}$ erases unit i, we add a request for pseudo block i to η (since $\mathcal{N}$ may erase many units at once, we impose an arbitrary order on these erasures). Now $\mathcal{N}_2$ runs the offline algorithm for the single-slot case on η. When the single-slot offline algorithm switches blocks among two units, $\mathcal{N}_2$ switches the corresponding actual units.

Theorem 9. *If $H \gg 1$ then the wear $h_i^{\mathcal{N}_2}(t)$ of any unit i at time $t = (1 - o(1))(2kHn/3)^{2/3}$ is at most H.*

The algorithm $\mathcal{N}_2$ is clearly not atomic. We can transform it into an atomic one in two ways.

Theorem 10. *There exist atomic algorithms $\mathcal{A}_1, \mathcal{A}_2$ such that for $H \gg 1$ the wear $h_i^{\mathcal{A}_1}(t_1), h_i^{\mathcal{A}_2}(t_2)$ of any unit i at times $t_1 = (1 - o(1))(2kHn/3)^{2/3}$ and $t_2 = \Omega\left((2kHn/3)^{2/3}/\log k\right)$ is at most H. $\mathcal{A}_1$ requires $k + 2$ empty units and $\mathcal{A}_2$ requires 3 empty units.*

The difficult non-atomic part of $\mathcal{N}$ is the sorting of the blocks in the dirty units. $\mathcal{A}_1$ uses a straightforward approach that requires $k + 1$ empty units to preform this sorting. $\mathcal{A}_2$ does it using the merge-sort algorithm, using two extra units to hold the partial runs of sorted blocks that the algorithm constructs. It is not hard to see that two extra units are always sufficient, and that the sorting algorithm performs $O(k \log k)$ erasures. The same idea can be used to trade off any number of extra units for better endurance using multiway merge-sort. Both algorithms apply the atomic single-slot unit wear leveling algorithm, which requires another extra unit.

3.2 The Deterministic Online Fractional Problem

The endurance of any deterministic algorithm for the fractional problem depends on the number of extra slots, $nk - m$, just like in the single-slot problem.

Theorem 11. *For every deterministic algorithm α (even if α is non-atomic) there exists a sequence σ such that $\ell_\alpha(\sigma) \leq (nk - m + 1)(H + 1)$. Furthermore, there exists a deterministic non-atomic algorithm that achieves $\ell(\sigma) \geq (nk-m+1)(H+1)$ and an atomic variant that achieves $\ell(\sigma) \geq ((n-1)k - m + 1)(H+1)$ using a single empty unit.*

3.3 The Randomized Online Fractional Problem

We now present a lower bound for randomized online algorithms for the fractional problem. The main ingredient that we analyze is the number of erasures preformed by the algorithm. The bound that we prove depends on the number of empty slots s. In particular, we require that $s < nk/2$. Otherwise, even a deterministic algorithm can achieve high endurance.

Theorem 12. *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 $s < nk/2$:*

- *If $s = n^{1-\epsilon}$ for some constant $0 < \epsilon < 1$ then there exists a constant $c > 0$ and σ such that $E[\ell_\alpha(\sigma)] < cHn$.*
- *If $\frac{n}{polylog(n)} \leq s \ll n \log n$ then there exists a constant $c > 0$ and σ such that*
$$E[\ell_\alpha(\sigma)] < cHn \cdot \frac{\log n}{\log(n \log n/s)} .$$
- *If $s = \Omega(n \log n)$ then there is a constant $c > 0$ and σ such that $E[\ell_\alpha(\sigma)] < cHs$.*

The proof's idea is to observe a random request sequence and split it into phases of size $2s$. Since there are only s empty slots, the algorithm must clean at least s slots during each phase. Because the requests are random, during each phase there is not likely to be a unit with many dirty slots. Thus, the algorithm will be forced to perform many erasures.

4 Conclusions

Our analysis shows that Ban's simple randomized algorithm [2] is nearly optimal (both in the competitive sense and in the absolute sense) for the single-slot wear-leveling problem. The competitive performance of deterministic algorithms for the same problem is poor, although many flash-based systems do use deterministic algorithms. The effectiveness of deterministic algorithms in practice is probably due to the fact that request sequences are oblivious to the online algorithm.

Our analysis of the fractional wear-leveling problem leads to two conclusions. First, the fact that offline algorithms outperform online algorithms implies that in practice, it is advantageous to try to cluster blocks according to the expected time of their next modification. Second, the analysis justifies the separation of the erasure-minimization policy from the wear-leveling policy.

References

1. Mahmud Assar, Siamack Nemazie, and Petro Estakhri. Flash memory mass storage architecture incorporation wear leveling technique. US patent 5,479,638, US patent 5,388,083 (slightly different title), and US patent 5,485,595 (slightly different title), all filed 1993, issued 1995/6, and assigned to Cirrus Logic, 1993.
2. Amir Ban. Wear leveling of static areas in flash memory. US patent 6,732,221, filed 2001, issued 2004, and assigned to M-Systems, 2001.
3. Avraham Ben-Aroya. Competitive analysis of flash-memory algorithms. Master's thesis, School of Computer Science, Tel-Aviv University, April 2006. Available online at `www.tau.ac.il/~abrhambe`.
4. 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. US patent 6,000,006, December 1999. Filed August 25, 1997; Issued December 7, 1999; Assigned to BIT Microsystems.
5. 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.
6. 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.
7. 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. US patent 5,835,935, 1998. Filed September 13, 1995; Issued November 10, 1998; Assigned to Lexar Media.
8. Eran Gal and Sivan Toledo. Algorithms and data structures for flash memories. *ACM Computing Surveys*, 37:138–163, 2005.
9. Sang-Wook Han. Flash memory wear leveling system and method. US patent 6,016,275, January 2000. Filed November 4, 1998; Issued January 18, 2000; Assigned to LG Semiconductors.
10. Edwin Jou and James H. Jeppesen III. Flash memory wear leveling system providing immediate direct access to microprocessor. US patent 5,568,423, October 1996. Filed April 14, 1995; Issued October 22, 1996; Assigned to Unisys.
11. 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.
12. Karl M. J. Lofgren, Robert D. Norman, Gregory B. Thelin, and Anil Gupta. Wear leveling techniques for flash EEPROM systems. US patent 6,081,447 and US patent 6,594,183, filed 1998/1999, issued 2000/2003, and assigned to Western Digital and Sandisk, 1998.
13. Steven E. Wells. Method for wear leveling in a flash EEPROM memory. US patent 5,341,339, 1994. Filed November 1, 1993; Issued August 23, 1994; Assigned to Intel.