Information Processing Letters 166 (2021) 106042

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

A competitive analysis for the Start-Gap algorithm for online memory wear leveling

William E. Devanny[a], Michael T. Goodrich[b], Sandy Irani[b,*]

[a] *Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210-1210 USA*
[b] *Department of Computer Science, University of California, Irvine, CA 92697 USA*

ARTICLE INFO

Article history:
Received 8 April 2016
Received in revised form 27 May 2020
Accepted 17 October 2020
Available online 13 November 2020
Communicated by Marek Chrobak

Keywords:
Analysis of algorithms
Data structures
On-line algorithms

ABSTRACT

Erase-limited memory, such as flash memory and phase change memory (PCM), has limitations on the number of times that any memory cell can be erased. The Start-Gap algorithm has shown a significant ability in practice to distribute updates across the cells of an erase-limited memory, but it has heretofore not been characterized in terms of its competitive ratio against an optimal offline algorithm that is given all the update addresses in advance. In this paper, we present a competitive analysis for the Start-Gap wear-leveling algorithm, showing that under reasonable assumptions about the sequence of update operations, the Start-Gap algorithm has a competitive ratio of $1/(1-o(1))$.

1. Introduction

Flash memory and phase change memory (PCM) are growing in popularity and are often included in computing devices sold today, due to their speed, power-consumption, and persistence characteristics. For instance, common installations configure fashion flash memory as a solid state drive and PCM as a main memory (e.g., see [1–3]), although flash memory can also be configured as a main memory as well [4].

In spite of their advantages, however, these memories have a drawback—there is an implicit upper bound on the number of times that a cell in such memories can be erased and over-written (e.g., see [1–3]). For instance, flash memory cells are projected to gracefully endure only up to about 10^5 erasures and PCM cells are projected to gracefully endure only up to about 10^7 erasures. Because of this limitation, there has been considerable work on heuristics for *wear leveling*, that is, strategies for relocating objects in such erase-limited memories so as to spread the amount of cell erasures and rewriting that is done (e.g., see [5–7]), as well as strategies for reducing the wearing of erase-limited memories for specific kinds of data structures and algorithms (e.g., see [8,9]).

In this paper, we study wear leveling algorithmically. We model erase-limited memory as an indexable set of memory cells that have an upper-bound limit, L, on the number of times that any cell can be over-written during its lifetime, and we study a well-known online wear-leveling algorithm in an online setting. We analyze this online algorithm in terms of its *competitive ratio* (e.g., see [10]), that is, the ratio of the performance of our online algorithm with that of an optimal offline algorithm.

In particular, we utilize the approach of Ben-Aroya and Toledo [11] to define a memory model consisting of $N \geq 2$ cells, indexed from 0 to $N-1$, each of which is a single memory word or block, such that there is a known parameter, L, that specifies an upper-bound limit on the number of times that any cell of this memory can be rewritten. For instance, we may have $L = 10^5$ or $L = N^{1/2}$, both of which are well-motivated from the published limitations

* Corresponding author.
E-mail addresses: devanny.2@osu.edu (W.E. Devanny), goodrich@ics.uci.edu (M.T. Goodrich), irani@ics.uci.edu (S. Irani).

https://doi.org/10.1016/j.ipl.2020.106042
0020-0190/© 2020 Elsevier B.V. All rights reserved.

for flash memory and PCM. Also, because erase-limited memory may be used as a main memory, we assume here that we have only a small (possibly constant) number of memory words available, each consisting of $d \log N$ bits, for some constant, $d \geq 1$, such that each of these words can be rewritten an unlimited number of times. These erasure-unlimited memory cells could, for instance, represent the registers used by a CPU or a memory controller. We refer to this model as the *erase-limited memory* model, with limit L, or notationally as ELM_L. If the parameter L is understood from the context or is assumed to be at most polynomial in N, then we refer to this simply as the ELM model.

1.1. Related prior work

In terms of prior work on specific algorithms for erase-limited memory, Chen et al. [8] study the database problems of indexing and join methods in the context of optimizing the performance of a computer using PCM, and Eppstein et al. [9] study hashing with respect to erase-limited memory. Irani et al. [12] study several schemes for performing general and specific algorithms using write-once memories, in both a word-based write-once memory (*WOM*) model and a bit-wise write-once memory (*bit-WOM*) model. Thus, their results for the WOM model can be viewed as applying to the ELM_1 model (that is, $\mathrm{ELM}_1 =$ WOM), and their algorithms for the bit-WOM model can be directly simulated in the $\mathrm{ELM}_{\log N}$ model. They do not discuss how to efficiently perform computations for a memory that allows a bounded number of rewrites greater than 1, but they do address how an algorithm in what we are calling ELM_L could be simulated in the WOM model with an $O(\log L)$ overhead ratio with respect to running time, using an indexing method similar to one used by Vitter [13].

Ben-Aroya and Toledo [11] provide competitive analyses for several wear-leveling algorithms, but their methods depend on a lookup table of size $O(N)$ stored in regular memory to maintain an erase-limited memory of size N. Thus, their methods are not for the ELM model under our assumption of only a constant number of regular-memory registers. That is, their algorithms are not for applications needing a small real memory, such as using erase-limited memory for a main memory.

Qureshi et al. [7] overcome this limitation with an algorithm they named *Start-Gap*, which uses only two registers to spread updates across an erase-limited memory. In a nutshell, their algorithm initially shuffles memory according to a random permutation and then incrementally shifts elements according to a cyclic shift of this permutation, using these two registers, as memory updates occur. Qureshi et al. provide an experimental analysis of their Start-Gap algorithm in their paper, which as of this writing has been cited over 700 times according to Google Scholar, but they do not provide a competitive analysis. Furthermore, we are not aware of any prior work providing a non-trivial competitive analysis of the Start-Gap algorithm, although Barcelo et al. [14] provide a competitive analysis on the energy efficiency of several caching algorithms for PCM.

1.2. Our results

We start by showing a lower bound of $(N/2)(1-o(1))$ on the competitive ratio of any deterministic algorithm with no restrictions on the input sequence. In the rest of the paper, we present a novel competitive analysis of the Start-Gap wear-leveling algorithm, showing that under a reasonable assumption about the request sequence, with probability $1-o(1)$, the Start-Gap algorithm can serve $(1-o(1))NL$ write requests before attempting to write to a memory location more than L times. The analysis holds as long is $L = N^{\alpha}$, for any $0 < \alpha < 1$. Note that NL is the longest sequence that can be handled by any algorithm, even the optimal offline algorithm. Thus, our analysis implies a competitive ratio of $1/(1-o(1))$ for the Start-Gap algorithm, in spite of it using only a constant-sized cache of erasure-unlimited memory (in fact, just two registers). In consideration for this constant-sized cache and the unlikely nature of malicious write sequences, the assumption made on the request sequence in our analysis is that no single block is erased and overwritten more than a certain number of times, $\beta \in o(L^{1/2})$, within any contiguous subsequence of N writes. The Start-Gap algorithm needs to know β or an upper bound on the value of β in order to tune a parameter of the algorithm. We then examine the case in which there is a limited sized cache whose cells can be erased and overwritten to an unlimited number of times. We show that with such a modestly-sized cache, the request sequence can effectively be made to have the required property to guarantee $(1-o(1))LN$ successful writes, even under malicious write sequences.

2. A model for memory-controller algorithms

We begin our analysis with a formal definition of the model. We assume that a memory-controller algorithm must store M blocks of memory in N locations. At any given time, such an algorithm maintains a placement that is a one-to-one function, $p : [M] \to [N]$, where $p(i)$ is the location of block b_i and $[k]$ is the set of the first k non-negative integers. W.l.o.g., let us assume for the sake of our analysis that each location in memory has a counter that indicates the number of writes that have been done to that location, with the set of counters being represented as a function $c : [N] \to [L]$. In the physical model, each memory location can be erased at most a certain number of times, so the counter for a location is incremented every time the contents of that location are erased. The controller algorithm receives a series of requests to write to blocks. Upon a request to write to a block, b_i, the algorithm may select to leave all the blocks in their current locations, in which case $c(p(i))$ is incremented, as this choice implies the block at location $p(i)$ was erased and overwritten. Alternatively, the algorithm may choose to move blocks around, including the requested block. A block can be moved to any empty location. Moving a block from location i to an empty location j results in an increment performed on i's counter, as this implies location i is erased in order to become empty. Although it might be possible to just mark location i as empty and perform the erasure when we want to write a new memory block to that location, this type of "lazy" erasing only allows for at most one additional write operation per location. If the requested block is moved, we assume that the new value for the block is written in its new location, thus satisfying the request. The algorithm can also select to swap the blocks in two locations which results in an increment performed on the counter of each of the two locations. As soon as the algorithm receives a request to write to a block b_i such that $c(p(i)) = L$, it can no longer continue. The question asked by our competitive analysis is the following:

If requests are received in an online manner, how many requests can be handled by an online algorithm in comparison to the optimal controller?

Note that any algorithm, even an optimal offline algorithm, can serve at most LN write operations. Therefore, we consider only sequences of length LN and ask how long a prefix of the sequence can be served before the algorithm needs to write more than L times to any location.

Theorem 1. *If $M = N - 1$, then the competitive ratio of any deterministic online memory controller algorithm is at least $(N/2)(1 - o(1))$.*

Proof. If $M = N - 1$, then there is exactly one empty location. The adversary strategy will only request memory blocks in locations 0 or 1, which can be achieved for any deterministic algorithm known to the adversary. If either location 0 or 1 is empty, the adversary will request the block in the non-empty location. If locations 0 and 1 are both non-empty, then the adversary will make a request to the memory block in the location (0 or 1) with the highest count. Note that regardless of the decisions made by the algorithm, the sum of the counters for locations 0 and 1 increases with each request. Thus, one of the two counters must be at least $L+1$ by the time $2L+1$ requests have been seen. Therefore, the algorithm can serve at most $2L$ requests before a failure.

Now we need to argue that there is a long ($\approx NL$) sequence of requests that an optimal offline algorithm can successfully serve such that the first $2L + 1$ requests in the sequence are the requests generated by the adversary strategy above. An optimal algorithm can serve the first $2L + 1$ requests as follows: keep the memory blocks in a static placement until there is a request to a memory block whose counter is equal to $L - 1$, then move that memory block to the empty location. This method will avoid failure for the first $2L - 1$ requests. As long as $L > 5$, there will be at most two locations whose counters are $\geq L - 2$. Swap the memory blocks in those locations with any memory block that is not requested in the next two requests and whose counters are $\leq L - 3$. As long as $N > 5$, there will be two such locations for the swaps. No counter will exceed L in the next two requests. After the first $2L+1$ requests, the total sum of the counters will be at most $2L + 5$: $2L + 1$ for the first $2L + 1$ requests plus at most 4 for the two swaps. Fill out the remaining sequence of $NL - (2L + 5)$ requests by requesting any memory block whose counter is $\leq L$. Therefore, the optimal algorithm can serve at least $NL - 4$ requests total. The competitive ratio is $(NL - 4)/2L$ which is $(N/2)(1 - o(1))$. $\square$

Note that the above lower bound applies even to deterministic algorithms with sizable caches of erasure-unlimited memory. To get a better bound than $N/2$ on the competitive ratio for algorithms, such as the Start-Gap algorithm, that have bounded-size caches of erasure-unlimited memory, it is therefore necessary to consider some assumptions about the input sequence or to consider randomized online algorithms.

3. The Start-Gap algorithm

The Start-Gap algorithm assumes that the memory is not completely full (i.e., that $M \leq N - 1$). The algorithm is first described for the case where $M = N - 1$. We argue below that the algorithm can be extended to the situation where $M < N - 1$, and that the worst case for the analysis is that $M = N - 1$. Let π be a random permutation of $[N - 2]$ that can be evaluated without using erasure-limited memory. The *Start-Gap* algorithm is parameterized by an integer, k, and works as follows:

1. Initially each block b_i is placed in location $\pi(i) + 1$. A "gap" register l indicates the empty location. Because the empty location is initially 0, $l = 0$ initially.
2. Suppose that the empty location is location l. Every k requests (which is maintained with a "count" register), the algorithm moves the block in location $l + 1 \bmod N$ to location l. Location $l + 1 \bmod N$ is erased and becomes the new empty location (and the count register is reset).

This is clearly a simple algorithm, which Qureshi et al. [7] show works well in practice. Let us, therefore, analyze this algorithm in terms of its competitive ratio.

Note that if $M < N - 1$, then there is more than one "gap" register. The same algorithm can be implemented by keeping a pointer to one of the gap registers and advancing the pointer in the same manner every k requests. If the location that the gap moves to is already empty, then the counter for that location is not incremented. The analysis for the $M = N - 1$ case carries over to the $M < N - 1$ case by the following reasoning: suppose instead that there were in fact $N - 1$ blocks of memory but $N - M - 1$ of the blocks are dummy blocks that are never written. The behavior of the algorithm is the same for the M original blocks. The analysis for $M = N - 1$ would hold in this case and the number of requests served would be at most the number of requests served in the $M < N - 1$ case. Since the lower bound of NL for the optimal holds regardless of the value of M, the upper bound follows.

4. A competitive analysis of the Start-Gap algorithm

Consider the requests to be numbered 0 through $NL - 1$. Of course, the operating system needs a function, f, that maps a pair (i, t) to the location, l, such that block b_i is in location l at the tth request. Block b_i stays in location $\pi(i) + 1$ for the first $(\pi(i) + 1)k$ requests and then moves one location to the left every kN requests. Letting $l(i,t) = \lfloor (t - (\pi(i)+1)k)/(Nk) \rfloor$, then, at the tth request, block b_i is in location $f(i,t) = (\pi(i) - l(i,t) + 1) \bmod N$.

Each location j defines a partition of the requests into contiguous subsequences, which we call *epochs*, as follows. The first epoch is only a partial epoch and lasts for jk requests. Thereafter all epochs (except possibly the last) are full epochs and last exactly kN requests. Each full epoch for location j starts when the location is vacated. After k requests, the block in location $j+1 \bmod N$ moves into location j and stays there for the remaining $k(N-1)$ requests of the epoch. Define $r = \lfloor (t - jk)/Nk \rfloor$. The block in location j at the tth request is block $\pi^{-1}((j+r) \bmod (N-1))$.

Definition 2. Consider a sequence of write requests. Divide the sequence into subsequences of N consecutive requests where the requests in a subsequence range from iN through $(i+1)N-1$. The sequence is said to be β-**balanced** if there are at most β writes to any particular block within a subsequence.

Note that the trivial static algorithm that never moves the blocks of memory is β-competitive when the input is restricted to β-balanced request sequences. This holds because no memory location is written more than β times in a block of N requests, which means that there will be at least L/β blocks of N requests before the static algorithm fails. The competitive ratio of $1/(1-o(1))$ for Start-Gap is considerably better for any $\beta > 1$. Thus, the static algorithm only does as well as Start-Gap for the extremely restrictive case of 1-balanced request sequences.

Lemma 3. *For any $0 < \epsilon < 1$, consider a sequence of requests that is β-balanced for*

$$\beta \le \sqrt{\frac{\epsilon^2 L}{2k \log N}}.$$

With probability at least $1 - 1/N$, the Start-Gap algorithm with parameter k can serve at least T requests without exceeding the write capacity of any location, for

$$T = \left(1 - \epsilon - \frac{2k}{L} - \frac{(1-\epsilon)}{k} - \frac{L}{kN}\right) LN.$$

Proof. Define $\gamma = L/kN$. Note that if $\gamma \ge 1$, the lemma is vacuously true, so we will assume that $\gamma < 1$. $T = (1 - \epsilon - (1-\epsilon)/k - \gamma)LN - 2kN$. For any location, the number of epochs, e, in the first T requests is at most $(1 - \epsilon - (1-\epsilon)/k - \gamma)L/k$, including the partial epochs at the beginning and end of the sequence. Since we are upper bounding the number of writes to a given location over the course of the request sequence, we can assume the worst case, which is that the beginning and ending epochs last for a full kN requests. Since $1 > 1 - \epsilon > 0$,

$$\begin{aligned} e &< \frac{L}{k}\left(1 - \epsilon - \frac{(1-\epsilon)}{k} - \gamma\right) \le \frac{L}{k}(1-\epsilon)\left(\frac{k-1}{k} - \gamma\right) \\ &\le \frac{L}{k}(1-\epsilon)\left(\frac{k-1}{k}\right)(1-\gamma). \end{aligned} \tag{1}$$

For any $\gamma > 0$, $(1-\gamma) \le 1/(1+\gamma)$. Therefore,

$$\begin{aligned} e &< \frac{L}{k}(1-\epsilon)\left(\frac{k-1}{k}\right)(1-\gamma) \\ &\le \frac{L}{k}(1-\epsilon)\left(\frac{k-1}{k}\right)\left(\frac{1}{1+\gamma}\right). \end{aligned} \tag{2}$$

Select an arbitrary location, l. Define X_i to be a random variable denoting the total increase to the counter for l during l's ith epoch with i ranging from 1 to e. The epoch begins by vacating location l which causes $c(l)$ to increase by 1. During the first k requests, the location l is empty. After that, there are the writes caused by the requests to the block that moved into location l. Note that there may be a slight dependence between the X_i's. Since the original permutation assigning blocks to locations was random, we know that the block sitting in the location l for its ith epoch is randomly chosen, except that it cannot be one of the $i-1$ blocks that were in location l during the first $i-1$ epochs. Suppose that t of the remaining $k(N-1)$ requests are to blocks that occupied location l some time during its first $i-1$ epochs. These requests do not add to X_i since these blocks are not in location l during epoch i. For any of the other $k(N-1)-t$ requests, the probability that the requested block is in location l is $1/(N-1-(i-1))$. The worst case is $t = 0$. Thus, we have that

$$\mathbf{E}[X_i \mid X_1, \ldots, X_{i-1}] \le 1 + \frac{k(N-1)}{N-(i-1)-1} \le 1 + \frac{kN}{N-i}$$

Using the fact that $i \le e$ and the upper bound for e from Equation (2): $e \le N\gamma/(1+\gamma)$:

$$1 + \frac{kN}{N-i} \le 1 + k(1+\gamma) \le (k+1)(1+\gamma).$$

We define a supermartingale with $Y_i = \sum_{j=1}^{i}(X_j - (k+1)(1+\gamma))$. Note that $\mathbf{E}[Y_i \mid Y_1, \ldots, Y_{i-1}] \le 0$. Also, $Y_i - Y_{i-1} = X_i \le \beta k$, since the request sequence is β-balanced. We will show below that if $\sum_{j=1}^{e} X_j > L$, then $Y_e > \epsilon L$.

By the Azuma-Hoeffding inequality [15], we have that

$$\Pr[Y_e > \epsilon L] \le \exp\left(-\epsilon^2 L^2 / 2e(\beta k)^2\right).$$

Using the fact that $e \le L/k$ and $\beta^2 \le \epsilon^2 L/2k \log N$, it follows that $\Pr[Y_e \ge \epsilon L] \le 1/N^2$. Thus, the probability that the counter for a particular location increases past L is bounded by $1/N^2$. The probability that the counter for any location increases beyond L is at most $1/N$. Therefore the probability that the sequence of requests can be served is at least $1 - 1/N$.

Now we show that if $\sum_{j=1}^{e} X_j > L$, then $Y_e \ge \epsilon L$. $Y_e = \sum_{j=1}^{e} X_j - e(k+1)(1+\gamma)$. If $\sum_{j=1}^{e} X_j > L$, then $Y_e > L - e(k+1)(1+\gamma)$. Using the upper bound for e from (2):

$$\begin{aligned} Y_e &> L - (k+1)(1+\gamma)\frac{L}{k}(1-\epsilon)\left(\frac{k-1}{k}\right)\left(\frac{1}{1+\gamma}\right) \\ &= L - L(1-\epsilon)\frac{(k+1)(k-1)}{k^2} \\ &> L(1-(1-\epsilon)) = \epsilon L \quad \square \end{aligned}$$

This implies the following.

Theorem 4. *Consider an erase-limited memory with $L = N^\alpha$ for $0 < \alpha < 1$. If a request sequence is β-balanced for $\beta = O(L^{1/2-\delta/2})$ and $0 < \delta < 1$, then the Start-Gap algorithm can serve at least the $(1 - o(1))NL$ first requests in the sequence with probability $1 - o(1)$.*

Proof. Select $\epsilon = L^{-\delta/4}$ and $k = L^{\delta/4}$ and use the bound from the lemma to show the following:

$$\beta = O(L^{1/2-\delta/2}) = o\left(\sqrt{\frac{\epsilon^2 L}{2k \log N}}\right).$$

With these parameters, $T = (1 - o(1))LN$. $\square$

We say that a probability distribution over a request sequence is *p-bounded* if each request for a block is generated independently according to a distribution in which the probability that any particular block is requested is bounded by p. Note that the access distribution over the blocks can change over time but it cannot depend on past requests.

Corollary 5. *Consider an erase-limited memory with $L = N^\epsilon$ for $0 < \epsilon < 1$. If a request sequence is generated by a p-bounded probability distribution for $p = O(L^{1/2-\delta}/N)$ and $0 < \delta < 1/2$, then the Start-Gap algorithm can serve at least $(1 - o(1))NL$ requests in the sequence with probability $1 - o(1)$.*

Proof. We show that if a request sequence of length LN is generated according to a p-bounded distribution with $p \leq L^{1/2-\delta}/N$, then with probability $1 - o(1)$ it is β-balanced for $\beta = L^{(1-\delta)/2}$. The corollary then follows from Theorem 4.

Divide the request sequence into L subsequences of N consecutive requests. Select one particular subsequence and one particular block b. We have an indicator variable X_i, which is 1 if b is requested for the ith request in the subsequence and 0 otherwise. The total number of requests to b during the subsequence is $X = \sum_{i=1}^N X_i$. The X_i are i.i.d. with expectation at most $p = L^{1/2-\delta}/N$. $\mu = L^{1/2-\delta}$ is an upper bound on the expected value of X.

Using a Chernoff inequality,

$$\Pr[X \geq (1+\gamma)\mu] \leq \exp(-\gamma^2\mu/3).$$

We would like to bound the probability that $X \geq L^{(1-\delta)/2}$. Setting, $\gamma = L^{\delta/2} - 1$, we get that

$$\Pr[X \geq L^{(1-\delta)/2}] \leq \exp(-(L^{\delta/2} - 1)^2 L^{1/2-\delta}/3).$$

For $L = N^\epsilon$, $\Pr[X \geq L^{(1-\delta)/2}]$ is $O(\exp(-N^{\epsilon'}))$ for some ϵ'. The probability that any of the N blocks is requested more than β times during any of the L subsequences is at most $NL\exp(-N^\epsilon/3)$, which is $o(1)$. $\square$

5. A variation—the ELM model with a cache

The above results address the likely common situation in which the distribution of accesses (that is, writes to memory) can have significant variance but does not contain any high-frequency accesses to the same location, as might occur in a malicious attack on an erase-limited memory [16]. Now we show that even such malicious access sequences can be managed if we consider the ELM Model augmented to have a cache of regular memory that can hold a limited number of blocks and can be written to an unlimited number of times.

At any point in time, cache-augmented Start-Gap can choose to copy a block into the cache. The block will be replaced in the ELM by a dummy copy that will move along from location to location in the ELM according to the rules of the standard Start-Gap algorithm. Any writes performed on a block stored in the cache are performed only on the cached copy of the block. When the block is evicted from the cache, it is written to the location of its dummy version.

We propose a simple cache administration policy, which has a competitive ratio that is surprisingly efficient. The decision about whether to move a requested block into the cache is purely random: copy the requested block into the cache with probability p for some fixed p. The cache will use a version of LRU called p-LRU as a replacement policy. The algorithm LRU (which stands for Least-Recently-Used) maintains a queue of the blocks in the cache. When a block is requested that is already in the cache, the block is moved to the tail of the queue. When a new block is brought into the cache, the block at the head of the queue is evicted and the new block is inserted at the tail of the queue. The only difference between p-LRU and LRU is that on a request to a block already in the cache, p-LRU moves the requested block to the tail of the queue with probability p (instead of with probability 1). The value for p is chosen to be $C/2N$, where C is the number of blocks that can fit in the cache. We then have the following result.

Theorem 6. *If $L = N^\epsilon$ for $0 < \epsilon < 1$, and $C \geq N/L^{1/2-\delta}$ for $0 < \delta < 1$, then with probability $1 - o(1)$, the Start-Gap Algorithm with a random cache policy will be able to service $(1 - o(1))LN$ write requests.*

Proof. Divide the request sequence into L subsequences of N consecutive requests. We call a write request an *outside* request if the requested block resides outside the cache (and therefore causes an additional write to its ELM location). We show that the probability that there are more than $L^{(1-\delta)/2}$ outside requests to a block in a single subsequence before it is brought into the cache is $o(1)$. We also show that the probability that a block is evicted from the cache in a subsequence in which it was requested is $o(1)$. This means that with probability $1 - o(1)$, there are never more than $L^{(1-\delta)/2}$ outside requests to a block in a single subsequence. The theorem then follows from Theorem 4.

Now we determine the probability that any block outside the cache is requested more than $L^{(1-\delta)/2}$ times in any sequence without being cached. Select a block b and a particular subsequence. Let m be the number of times b is requested in the subsequence. If b starts out the subsequence in the cache, start counting requests to b only after it is evicted. If $m \leq L^{(1-\delta)/2}$, then b is certainly not requested more than $L^{(1-\delta)/2}$ times outside the cache. Now suppose that $m > L^{(1-\delta)/2}$. The probability that on any of these m requests b is brought into the cache is $p = C/2N = L^{1/2-\delta}/2$. The probability that b is not cached during the first $L^{(1-\delta)/2}$ requests is

$$(1-p)^{L^{(1-\delta)/2}} = \left(1 - \frac{1}{2L^{1/2-\delta}}\right)^{L^{(1-\delta)/2}}$$
$$= O(\exp(-L^{\delta/2}/2)).$$

The probability that any block is not cached during its first $L^{(1-\delta)/2}$ outside requests in any subsequence is at most $LN \exp\left(-L^{\delta/2}/2\right)$. For $L = N^{\epsilon}$, the probability is $o(1)$.

Now we determine the probability that a block b is evicted from the cache during the same subsequence in which it is brought into the cache. In order for the eviction to happen, there have to be at least C requests in which the requested block is moved to the tail of the LRU queue. The block can come from the ELM or from the cache itself. Either way, each request results in a block being moved to the tail of the queue independently with probability $p = C/2N$. Let X_i be the indicator variable denoting whether the ith requested block in the subsequence is moved to the tail of the LRU queue. Let X denote the sum of the X_i: $X = \sum_{i=1}^{n} X_i$. The expectation of X is $\mu = C/2$. By a Chernoff bound,

$$\Pr[X \geq (1+\gamma)\mu] \leq \exp(-\gamma^2\mu/3).$$

Using $\gamma = 1$,

$$\Pr[X \geq C] = \Pr[X \geq 2\mu] \leq \exp(-\mu/3)$$
$$= \exp\left(\frac{-N}{6L^{1/2-\delta}}\right).$$

The probability that any block is evicted from the cache in any subsequence in which it is requested is at most $NL\exp(-N/6L^{1/2-\delta})$, which is $o(1)$ for $L = N^{\epsilon}$, $0 < \epsilon < 1$. □

Declaration of competing interest

The authors certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Acknowledgements

This research was supported in part by the National Science Foundation under grant 1815073.

References

[1] R. Bez, E. Camerlenghi, A. Modelli, A. Visconti, Introduction to flash memory, Proc. IEEE 91 (4) (2003) 489–502.
[2] P. Pavan, R. Bez, P. Olivo, E. Zanoni, Flash memory cells—an overview, Proc. IEEE 85 (8) (1997) 1248–1271.
[3] H.-S. Wong, S. Raoux, S. Kim, J. Liang, J.P. Reifenberg, B. Rajendran, M. Asheghi, K.E. Goodson, Phase change memory, Proc. IEEE 98 (12) (2010) 2201–2227.
[4] M. Wu, W. Zwaenepoel, eNVy: a non-volatile, main memory storage system, in: 6th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS), ACM, 1994, pp. 86–97.
[5] L.-P. Chang, On efficient wear leveling for large-scale flash-memory storage systems, in: ACM Symp. on Applied Computing (SAC), 2007, pp. 1126–1130.
[6] Y.-H. Chang, J.-W. Hsieh, T.-W. Kuo, Improving flash wear-leveling by proactively moving static data, IEEE Trans. Comput. 59 (1) (2010) 53–65.
[7] M.K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, B. Abali, Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling, in: 42nd IEEE/ACM Int. Symp. on Microarchitecture (MICRO), 2009, pp. 14–23.
[8] S. Chen, P.B. Gibbons, S. Nath, Rethinking database algorithms for phase change memory, in: 5th Conf. on Innovative Data Systems Research (CIDR), 2011, pp. 21–31.
[9] D. Eppstein, M.T. Goodrich, M. Mitzenmacher, P. Pszona, Wear minimization for cuckoo hashing: how not to throw a lot of eggs into one basket, in: J. Gudmundsson, J. Katajainen (Eds.), 13th Int. Symp. on Experimental Algorithms (SEA), in: LNCS, vol. 8504, Springer, Cham, 2014, pp. 162–173.
[10] S. Albers, Online algorithms: a survey, Math. Program. 97 (1) (2003) 3–26.
[11] A. Ben-Aroya, S. Toledo, Competitive analysis of flash-memory algorithms, in: Y. Azar, T. Erlebach (Eds.), European Symp. on Algorithms (ESA), in: LNCS, vol. 4168, Springer, 2006, pp. 100–111.
[12] S. Irani, M. Naor, R. Rubinfeld, On the time and space complexity of computation using write-once memory or is pen really much worse than pencil?, Math. Syst. Theory 25 (2) (1992) 141–159.
[13] J.S. Vitter, An efficient I/O interface for optical disks, ACM Trans. Database Syst. 10 (2) (1985) 129–162.
[14] N. Barcelo, M. Zhou, D. Cole, M. Nugent, K. Pruhs, Energy efficient caching for phase-change memory, in: G. Even, D. Rawitz (Eds.), 1st Mediterranean Conf. on Algorithms (MedAlg), in: LNCS, vol. 7659, Springer, 2012, pp. 67–81.
[15] B. Bercu, B. Delyon, E. Rio, Concentration Inequalities for Sums and Martingales, Springer, 2015.
[16] A. Seznec, A phase change memory as a secure main memory, Comput. Archit. Lett. 9 (1) (2010) 5–8.