Randomized Weighted Caching with Two Page Weights

Sandy Irani

April 18, 2001

Abstract

We consider a special case of the weighted caching problem where the weight of every page is either 1 or some fixed number $M > 1$. We present a randomized algorithm which achieves a competitive ratio which is $O(\log k)$ where k is the number of pages which can fit in the cache.

1 Introduction

A caching algorithm manages a two-level store consisting of a fast memory (or *cache*) that can hold k pages, and a slow memory. The algorithm is presented with a sequence of requests to virtual memory pages. If the page requested is in fast memory (a *hit*) it incurs no cost; but if not (a *fault*), the algorithm must bring it in to fast memory at unit cost, and decide which of the k pages currently in fast memory to evict in order to make room for it. In the weighted version of the caching problem, each page p has a non-negative weight $w(p)$. The cost of bringing the page p into the cache is $w(p)$. Note that although pages have different weights, they all have the same size, so it is always the case that exactly k pages can fit in the cache at any point in time. We will think of the cache as having k *slots* each of which can hold a single page at any point in time. The cost incurred by an algorithm A on a sequence of requests σ is the sum of the weights of the requested pages for all requests on which the algorithm faults. We will denote this value by $\texttt{cost}_A(\sigma)$.

A caching algorithm is *online* if it makes the decision about which page to evict without knowledge of future requests. We evaluate the performance of an online algorithm with respect to the performance of the optimal offline algorithm. For a finite c, we say that A is a *c-competitive algorithm* if for all σ, $\texttt{cost}_A(\sigma) - c \cdot \texttt{cost}_{OPT}(\sigma)$ remains bounded by a constant. For a randomized algorithm A, we replace $\texttt{cost}_A(\sigma)$ by its expectation in the above definition. The *competitiveness* of A, denoted c_A, is the infimum of c such that A is c-competitive, if such a c exists. If no such c exists then A is said to be non-competitive; all algorithms discussed in this paper are competitive.

In this paper, we present and analyze a randomized algorithm for a special case of the weighted caching problem. We consider the problem where all requests are limited to pages whose weights are either 1 or some fixed integer M greater than 1. There are no other restrictions placed on M. For example, it could be the case that $M >> k$.

2 Previous Work

The competitive analysis of paging where the pages have uniform weight was first studied by Sleator and Tarjan [ST85]. They showed that any deterministic algorithm must have a competitive ratio of at least k. Furthermore, the well-known Least-Recently-Used algorithm achieves a competitive ratio of k. Randomized uniform paging was first studied by Fiat *et al.* [FKL$^+$91] who developed the Randomized Marking Algorithm and showed that it achieves a competitive ratio no worse than $2H_k$. Moreover, Fiat *et al.* showed a lower bound of $\ln k$ for the competitive ratio of any randomized algorithm. McGeoch and Sleator showed an algorithm which achieves a competitive ratio of $\ln k$ [MS91]. Later, Achlioptas *et al.* [ACN00] showed a much simpler algorithm achieving a competitive ratio of $\ln k$ and determined the exact competitive ratio of the Marking algorithm, $2H_k - 1$.

k-competitive deterministic algorithms for weighted caching have been known for some time [CKPV91, You94]. It has been conjectured [Fia01] that algorithms similar to the one presented here are $O(\log k)$-competitive ratio for the special case where pages have one of two fixed weights, but no bounds were known on their performance. Ours is the first algorithm to be proven $O(\log k)$-competitive for this problem, thus giving the first improvement over the bound for deterministic algorithms. Our algorithm, devised independently of the work in [Fia01], draws heavily on the Randomized Marking Algorithm introduced by Fiat *et al.* [FKL$^+$91]. It is essentially a nested version of the Randomized Marking Algorithm: the Randomized Marking Algorithm is run on each class of pages separately while making sure that each class of pages occupies a 'fair' share of the cache. We believe that this algorithm could, in principle, be extended to a constant number of classes of pages, although that question is not addressed in this paper. However, the constant factor in the competitive ratio is likely to increase rapidly with the number of classes which would limit extending this method to an arbitrary number of classes.

The weighted caching problem is also an important special case of the well-known k-server problem [MMS90]. In the k-server problem, k mobile servers move about on nodes in a metric space. The cost of moving a server from node a to node b in the metric space is the distance from a to b. A sequence of requests is presented to the algorithm. Each request is a node in the metric space. When a request is received, a server must be moved to that node. The goal is to serve a sequence of requests so as to minimize the total distance traveled by all servers.

The weighted cache problem is a special case of the k-server problem where the metric space has the property that each node has a weight and the distance to a node p from any other node in the metric space is the weight of p. The set of nodes represent virtual memory pages. The k mobile servers represent slots for pages in the cache. A server occupies a node whenever that page resides in the cache. The process of moving a server from a node p to a node q represents evicting p from the cache and replacing it by q. This move costs the algorithm $w(q)$.

Despite much effort, very little is known about randomized algorithms for the k-server problem. Bartal *et al.* have shown in [BBBT97] a randomized algorithm which achieves a competitive ratio which is polylogarithmic in k when the number of servers is one less than the number of nodes in the metric space (the *metric task system problem*). Also, Bartal *et al.* have shown an algorithm for two servers on the line [BCL98]. Other than these results, no other results for the randomized k-server problem are known which beat deterministic bounds. The result in this paper represents an important step towards randomized weighted caching and the randomized k-server problem.

3 The algorithm

We examine a specialization of the weighted caching problem where the weight of each page is either 1 or M. A page of weight 1 is said to be a *1-page*. A page of weight M is said to be an *M-page*. We will assume that M is an integer. If this is not the case, we can round M down to the nearest integer and execute the algorithm for the rounded value. This change will effect the competitive ratio by at most a constant factor.

Since the algorithm presented here is so heavily based on the Randomized Marking Algorithm (RMA) of Fiat *et al.* , we will review RMA here and point out some similarities. RMA falls into a general class of algorithms called marking algorithms first introduced by Karlin *et al.* In a marking algorithm, the set of pages in the cache are divided into marked and unmarked pages. Initially, all pages are unmarked. If the requested page is not already in the cache, then an unmarked page is evicted and the newly requested page is brought into the cache. If there are no unmarked pages in the cache on a fault, then all pages become unmarked before a page is chosen for eviction.

After the newly requested page is brought into the cache, it is marked. This process divides the request sequence into phases, where a new phase begins whenever the set of pages in the cache are unmarked. The Randomized Marking Algorithm is a marking algorithm which, on a fault, chooses an unmarked page uniformly at random for eviction.

The algorithm RandCache is shown in Figures 1, 2 and 3. At a high level, the algorithm presented here runs RMA on each class of pages separately, while making sure that each class of pages occupies an appropriate share of the cache. To determine how the cache is divided between the 1-pages and the M-pages the following set of rules are observed:

- All pages are marked as soon as they enter the cache.
- On a fault, if there are unmarked pages of the same class as the requested page, the algorithm picks one such page at random and evicts it.
- On a fault, if there are only unmarked pages of the other class,
  - If the requested page is an M-page, evict a random unmarked 1-page.
  - If the requested page is a 1-page,
    * Roughly, every M^{th} time this happens, evict a random unmarked M-page. This is achieved using the counter N_1 in Figure 1.
    * All other times, unmark all 1-pages and evict one such page at random.
- If on a fault, all pages are marked,
  - Unmark all 1-pages.
  - Unmark all M-pages pages only if there have been M occasions where a 1-page did not replace an unmarked M-page. This is achieved using the counter N_2 in Figure 1.

The Figures 1,2 and 3 represent what is done when a request to a page p arrives. All marked pages reside in the cache. We start off with all pages unmarked. Furthermore, when the sequence starts, we assume that the cache is empty. To make the algorithm well defined, we will assume that the cache is actually filled with unmarked 1-pages which will never be requested. The algorithm makes use of two counters, N_1 and N_2, which are initialized to 0.

The marking scheme divides the sequence into a number of phases which are in turn divided into a number of subphases. A new phase begins whenever the procedure NewPhase is called. A new subphase begins whenever the procedure NewSubPhase is called. Specifically, if NewSubPhase(P) is called, the old subphase ends and a new one begins *before* the current request to page p.

We will work through the following two examples to illustrate how the algorithm works. Consider the case where $M = 2$ and $k = 4$. The M-pages are $\{A, B, C, D\}$ and the 1-pages are $\{1, 2, 3, 4\}$. The first sequence we will consider is $\sigma = 1, 2, 3, A, B, 4, C, 1, 2, 3$. For the first three requests, the requested page is brought into the cache and marked in step (16). For the fourth request, page A is brought into the cache and marked in step (7). At this point, the contents of the cache are $\{1, 2, 3, A\}$ and all pages in the cache are marked. When B is requested, the algorithm increments N_2 in step (10). Note that the algorithm starts with only 1-pages in the cache, so no M-pages have been evicted. Since $N_2 = 1$ and $M = 2$, a new subphase begins. The previous subphase consisted of requests $1, 2, 3, A$. Pages 1, 2 and 3 are unmarked. Then RandCache(B) is called again from NewSubPhase(B) and a randomly chosen page from 1, 2, or 3 is evicted. B is marked and brought into the cache in step (7). Let's say it was page 2 that was evicted. At this point, the contents of the cache are $\{1, 3, A, B\}$ and the pages A and B are marked. When 4 is requested, a randomly chosen page from 1 or 3 is evicted. 4 is marked and brought into the cache in step (16). Let's say it was page 1 that was evicted. When C is requested, page 3 is evicted and page C is marked and brought into the cache in step (7). Now the the contents of the cache are $\{A, B, C, 4\}$ and all pages in the cache are marked. When page 1 is requested, N_2 is incremented in step (28). Now $N_2 = M$ and NewPhase is called. The last subphase consisted of requests $B, 4, C$. The phase consisted of requests $1, 2, 3, A, B, 4, C$. At this point everything is unmarked. When RandCache(1) is called from NewSubPhase(1), page 4 is evicted and page 1 is marked and brought into the cache. Since page 1 was not requested in the previous subphase, N_1 is incremented in step (18). The contents of the cache are now $\{A, B, C, 1\}$ and only page 1 is marked. Now 2 is requested. Since $N_1 \leq M$, NewSubPhase(2) is called in step (25). The last subphases consisted only of the request to page 1. Page 1 is unmarked. When RandCache(2) is called from NewSubPhase(2), page 1 is evicted and page 2 is marked and brought into the cache. N_1 is incremented in step (18). The contents of the cache are $\{A, B, 1, 2\}$ and pages 1 and 2 are marked. Now 3 is requested. At this point $N_1 \geq M$. A randomly chosen page from A or B is evicted in step (21). Page 3 is marked and brought into the cache. N_1 is incremented by 1 in step (23) and decremented by 2 in step (24).

In the second example, the request sequence is $\sigma = A, B, C, D, 1$. For the first four requests, the requested page is marked and brought into the cache in step (7). Now 1 is requested. N_2 is incremented in step (28) but is still less than M, so NEWSUBPHASE(1) is called which in turn calls RANDCACHE(1). The last subphase consisted of requests A, B, C, D. Since there are no 1-pages in the cache, nothing has changed. N_2 is incremented again in step (28) and this time NEWPHASE(1) is called. Note that this results in an empty subphase. At this point all four M-pages in the cache are unmarked. Since there are no 1-pages in the cache, a randomly chosen M-page is evicted in step (21). Page 1 is marked and brought into the cache. N_1 is incremented by one in step (23) and then decremented by 2 in step (24). At this point $N_1 = -1$.

The main result of this paper is the following theorem:

Theorem 1 *The competitive ratio of the algorithm RandCache is* $O(\log k)$.

We start with a few important observations about the algorithm. Once a 1-page is brought into the cache, it remains marked and does not leave the cache until the end of the current subphase at which point all 1-pages become unmarked. This follows from the fact that the algorithm never evicts a marked page. The only place in the algorithm where 1-pages are unmarked is in NEWSUBPHASE which is only called when all 1-pages in the cache are marked. Thus, the set of distinct 1-pages requested in a subphase is the same as the set of marked 1-pages which RandCache has in its cache at the end of the subphase which is the same as the set of (marked or unmarked) 1-pages which RandCache has in its cache at the end of the subphase. Similarly, when an M-page is brought into the cache, it remains marked and in the cache until the end of the current phase at which point all pages become unmarked. This follows from the fact that the only place where M-pages are unmarked is in NEWPHASE which is only called when all pages in the cache are marked. Thus, the set of distinct M-pages requested in a phase is the same as the set of marked M-pages which RandCache has in its cache at the end of the phase which is the same as the set of (marked or unmarked) M-pages which RandCache has in its cache at the end of the phase.

Each subphase will be designated M-heavy or 1-heavy. The designation is determined as follows. At some point in the course of a phase it happens that there are no unmarked M-pages in the cache. This includes the case where there are no M-pages in the cache. Once this happens, there will be no unmarked M-pages in the cache for the remainder of the phase. Suppose that this first happens during the r^{th} subphase in the phase. If the phase starts with no M-pages in the cache, then let $r = 1$. All subphases after the r^{th} subphase are M-heavy. All subphases which precede the r^{th} subphase are 1-heavy. The r^{th} subphase is said to be 1-heavy if any M-pages are evicted to make room for a newly requested 1-page and is M-heavy otherwise.

We will use the following five lemmas in the proof of the main result. Throughout the proofs of these lemmas, the r^{th} subphase will be the subphase in which it first happens that there are no unmarked M-pages in the cache.

Lemma 2 *During a* 1*-heavy subphase, no request to an* M*-page is served by evicting a* 1*-page.*

Proof. We claim that whenever an M-page is brought into the cache during a 1-heavy subphase, there are unmarked M-pages in the cache. This means that the request will be served by evicting an unmarked M-page in step (5) of the algorithm. The claim is certainly true for all subphases before the r^{th} subphase since it is only in the r^{th} subphase that there are no unmarked M-pages in the cache. Moreover, by definition, the r^{th} subphase is M-heavy only if at some point t in that subphase, an M-heavy page was evicted to accommodate a request for a 1-page. This can only happen in step (21). When step (21) is reached, it must be the case that all 1-pages are marked and there are unmarked M-pages. Suppose this happens at time t. The point in time when there are no unmarked M-pages in the cache happens at some time t' which is greater than t. We know that t' happens within the subphase since the r^{th} subphase is when all the M-pages in the cache first become marked. Note that at time t' all pages in the cache are marked since all 1-pages were marked at time t. This means that any request after time t' for a page which is not in the cache will cause a new subphase to begin. Thus, we have established that if the r^{th} phase is 1-heavy, then the next fault after all M-pages become marked begins a new subphase. This means that whenever RandCache faults during the r^{th} subphase, there are unmarked M-pages in the cache. ■

Lemma 3 *During an* M*-heavy subphase, no request to a* 1*-page is served by evicting an* M*-page.*

Proof. The fact is certainly true for all subphases which follow the r^{th} subphase since all M-pages in the cache are marked and none of them will be evicted until the beginning of a new phase. The r^{th} phase is said to be M-heavy if and only if no request to a 1-page is served by evicting an M-page. The lemma follows. ■

Lemma 4 *There are* M M*-heavy subphases in a phase.*

Proof. A new phase will begin if and only if there are no unmarked pages in the cache and $N_2 \geq M$. N_2 starts out at 0. We need to establish that N_2 is incremented at the end of a subphase if and only if the subphase was

M-heavy. Note that N_2 is incremented at the end of a subphase if and only if there are no unmarked M-pages and no M-pages have been evicted in the subphase on a request to a 1-page. This means that at the end of the subphases which precede the r^{th} subphase (which are all 1-heavy), N_2 is not incremented because there are unmarked M-pages in the cache at the end of these subphases. For all the subphases after the r^{th} subphase (which are all M-heavy), all the M-pages are marked at the beginning of the subphase. Furthermore by Lemma 3, no M-pages are evicted in the subphase to accommodate a request for a 1-page. This means that N_2 is incremented.

Now we consider the r^{th} subphase. All M-pages are marked at the end of the r^{th} subphase. Furthermore, the phase is declared to be M-heavy if and only if no M-pages have been evicted to accommodate requests for 1-pages. Therefore N_2 is incremented if and only if the subphase is M-heavy. ■

For the next lemma, we will need the following definitions. A 1-page is said to be *new* if it is requested in the current subphase but was not requested in the previous subphase. An M-page is said to be *new* if it is requested in the current phase but was not requested in the previous phase.

Lemma 5 *Suppose that at the end of a* 1*-heavy subphase which is not the last* 1*-heavy subphase in the phase, there have been y new* 1*-pages requested in the phase so far. Then the number of M-pages which have been evicted to make room for a* 1*-page is at least $\lfloor y/M \rfloor$.*

Proof. First we must establish that N_1 is incremented exactly once for every new 1-page requested in the phase. This follows from the fact that just after a 1-page is brought into the cache, the algorithm checks if it is a new page. If it is a new page, N_1 is incremented by 1. Furthermore, these are the only times when N_1 is incremented.

Since it is a 1-heavy subphase and not the last 1-heavy subphase, the subphase ends with unmarked M-pages in the cache. Thus, any request to an M-page must be resolved in step (5) of the algorithm. This means that the subphase must end on a request to a 1-page in which there are unmarked M-pages left in the cache. This means the subphase ends in step (25) and it must be the case that $N_1 < M$.

The only time N_1 is decremented is when an M-page is evicted to make room for a 1-page in which case it is decremented by M. Putting this fact together with the fact that N_1 is incremented by 1 for every new 1-page requested and that N_1 starts at 0 and ends at a value below M, we get that if there have been y new pages requested in the subphase, then the number of times and M-page has been evicted to make room for a 1-page is at least $\lfloor y/M \rfloor$. ■

Lemma 6 *If there are $x > 1$ 1-pages served in a phase by evicting an M-page, then there are at least xM new 1-pages requested in that phase.*

Proof. By Lemmas 2 and 3, the number of 1-pages in the cache can not decrease during a 1-heavy subphase and can not increase in an M-heavy subphase. Since all the 1-heavy subphases precede all the M-heavy subphases, the number of 1-pages in the cache during the course of a phase is monotonically non-decreasing and then monotonically non-increasing. Thus, it can only happen once in a phase that the number of 1-pages in the cache goes from 0 to 1. Furthermore, in any phase, this can only happen the very first time an M-page is evicted to make room for a 1-page.

If it happens again in the phase that an M-page is evicted to make room for a 1-page, it must be the case that at this point, the number of 1-pages in the cache is at least 1. M-pages can only be evicted to make room for 1-pages in line (21) of Figure 1. If line (21) is reached and there is at least 1 1-page in the cache, then $N_1 \geq M$. In this case, when N_1 is decremented, it will remain non-negative. Thus, either it happens at most once that an M-page is evicted to make room for a 1-page or $N_1 \geq 0$ at the end of the phase. In the first case, the lemma holds vacuously. In the second case, observe that a new 1-page is requested for every time that N_1 is incremented and N_1 is decremented by M every time a 1-page is served by evicting an M-page. ■

We will fix a sequence σ and analyze the expected cost of RandCache and the cost of the optimal algorithm on σ. First, we require some definitions.

Suppose there are m phases in the sequence σ. For $1 \leq i \leq m$,

- Let t_i be the total number of distinct M-pages requested in the i^{th} phase.
- Let $t_{i,j}$ be the number of distinct 1-pages requested in the j^{th} subphase of the i^{th} phase.
- Let n_i be the number of M-pages requested in the i^{th} phase that were not requested in the $(i-1)^{st}$ phase (i.e., the number of new M-pages requested in phase i).
- Let $n_{i,j}$ be the number of 1-pages requested in the j^{th} subphase of the i^{th} phase that were not requested in the previous subphase (i.e., the number of new 1-pages in the j^{th} subphase of the i^{th} phase).
- Let $k_{M,i}$ be the number of slots in the cache of the optimal algorithm that ever hold an M-page in phase i or phase $i-1$. This includes the slots that hold an M-page at the beginning of phase $i-1$ and the slots into which an M-page is placed in phases $i-1$ or i.

- Let $k_{1,i}$ be the number of slots in the cache of the optimal algorithm that ever hold a 1-page in phase i or phase $i-1$. This includes the slots that hold a 1-page at the beginning of phase $i-1$ and the slots into which a 1-page is placed in phases $i-1$ or i.

RandCache has exactly $t_{i,j}$ 1-pages in the cache at the end of subphase j of phase i. Similarly, there are exactly t_i M-pages in the cache at the end of phase i. Suppose that there are p_i 1-heavy subphases in phase i. By Lemma 4, this means that there are a total of $p_i + M$ subphases in the i^{th} phase. At the end of phase i, there are t_i M-pages in the cache and t_{i,p_i+M} 1-pages in the cache. Thus, $t_i + t_{i,p_i+M} = k$.

By Lemma 2, during a 1-heavy subphase, no 1-pages are evicted to accommodate an M-page. Thus, we have that

$$t_{i-1,p_{i-1}+M} \leq t_{i,1} \leq t_{i,2} \leq \cdots \leq t_{i,p_i}.$$

During an M-heavy subphase, no M-pages are evicted to accommodate a 1-page. Thus, we have that

$$t_{i,p_i} \geq t_{i,p_i+1} \geq t_{i,p_i+2} \geq \cdots \geq t_{i,p_i+M}.$$

Theorem 1 follows from the following two lemmas.

Lemma 7

$$11\texttt{cost}_{OPT}(\sigma) \geq \sum_{i=1}^{m} \max\left\{M-2, \left[M \cdot n_i + \sum_{j=1}^{p_i+M} n_{i,j}\right]\right\} - (k+1)M.$$

Lemma 8

$$E[\texttt{cost}_{RC}(\sigma)] \leq 2\left(\left(2+\frac{1}{M}\right)\log k + 1\right)\sum_{i=1}^{m} \max\left\{M, \left[M \cdot n_i + \sum_{j=1}^{p_i+M} n_{i,j}\right]\right\} + k\log k.$$

We start with the proof of Lemma 7. We separate Lemma 7 into three bounds. The first bound will justify the 'max' part of the lower bound. Then we lower-bound the cost of the optimal algorithm by the sum of the weights of the new M-pages in each phase plus the cost of the new 1-pages in all of the M-heavy subphases. Finally we lower-bound the cost of the optimal algorithm by the sum of the weights of the new 1-pages in all the 1-heavy subphases.

Lemma 9 *Let S be the subset of $\{1,2,\ldots,m\}$ such that $i \in S$ if and only if $Mn_i + \sum_{j=1}^{p_i+M} n_{i,j} < M-2$. Then $\texttt{cost}_{OPT}(\sigma) \geq (|S|-1) \cdot M$.*

Lemma 10

$$6\texttt{cost}_{OPT}(\sigma) \geq \sum_{i=1}^{m} \left[M \cdot n_i + \sum_{j=p_i+1}^{p_i+M} n_{i,j} \right] - Mk.$$

Lemma 11

$$4\texttt{cost}_{OPT}(\sigma) \geq \sum_{i=1}^{m} \sum_{j=1}^{p_i} n_{i,j}.$$

Proof of Lemma 9. Consider a phase i such that $Mn_i + \sum_{j=1}^{M+p_i} n_{i,j} < M-2$. This means that $n_i = 0$ (i.e. there are no new M-pages requested) and the total number of new 1-pages requested is smaller than $M-2$. By the contrapositive of Lemma 6, there is at most one M-page evicted on a request to a 1-page in the phase. Thus, if there are y M-pages in the cache at the beginning of the phase, there are always at least $y-1$ M-pages in the cache. Furthermore, the number of M-pages in the cache never exceeds y (otherwise there would have to have been a new M-page requested in the phase).

We will first establish that there is an empty subphase among the last $M-1$ subphases. If a subphase is not empty, then there is a first served request either for a 1-page or an M-page. We will show that the first possibility can only happen at most $M-3$ times and the second possibility can happen at most once in the last $M-1$ subphases. Thus, it must be the case that one of the last $M-1$ subphases is empty.

Recall that the last $M-1$ subphases are all M-heavy subphases. Furthermore, at the beginning of each one of these subphases, all the M-pages in the cache are marked. If the first request of one of these subphases is to a 1-page, then it must be a new 1-page otherwise it would already be in the cache and no new subphase would have started. By the assumptions of the lemma, this happens at most $M-3$ times. If the first request of the subphase is for an M-page, then that page is not currently in RandCache's cache and the number of M-pages in the cache increases. The number of M-pages in the cache can not decrease during the last $M-1$ subphases since all the M-pages in the cache are marked. Thus, it can only happen once that the first request of one of the last $M-1$ subphases is for an M-page. Note that we are using the fact argued above that the number of M-pages in the cache never varies by more than one.

Thus, we have established that one of the last $M-1$ subphases is empty. This possibility occurs only when there are no unmarked 1-pages at the beginning of a subphase which means that there are no 1-pages in the cache. Since all the M-pages in the cache are marked (i.e. have been requested in the phase), this means that M distinct M-pages have been requested during the phase. A new page is then requested on the first request of the next phase. This means that from the interval after the first request of the current phase through the first request of the following phase, the optimal algorithm must evict an M-page. Thus, we have that for each phase i in which $Mn_i + \sum_{j=1}^{M+p_i} n_{i,j} < M - 2$, except for the last phase, the optimal algorithm must evict an M-page. ■

Proof of Lemma 10. We will denote the cost that the optimal algorithm incurs in serving requests for M-pages in σ by $\mathtt{cost}^M_{OPT}(\sigma)$.

In phases i and $i+1$, there are a total of $t_i + n_{i+1}$ distinct M-pages requested. These are served by the optimal algorithm using at most $k_{M,i+1}$ slots in the cache. Thus, during phase i and $i+1$, the optimal algorithm spends at least $M(t_i + n_{i+1} - k_{M,i+1})$ in serving M-pages. Thus, we can lower-bound the cost of the optimal algorithm by

$$\mathtt{cost}^M_{OPT}(\sigma) \geq M \sum_{i=0}^{\frac{m}{2}-1} (t_{2i+1} + n_{2i+2} - k_{M,2i+2}).$$

Since every algorithm starts out without any pages in the cache, the optimal algorithm must spend Mn_1 in the first phase. Thus, we have a lower bound of

$$\mathtt{cost}^M_{OPT}(\sigma) \geq Mn_1 + M \sum_{i=1}^{\frac{m}{2}-1} (t_{2i} + n_{2i+1} - k_{M,2i+1}).$$

Adding the two inequalities, we get that

$$2\mathtt{cost}^M_{OPT}(\sigma) \geq Mn_1 + M \sum_{i=1}^{m-1} (t_i + n_{i+1} - k_{M,i+1}).$$

Now we will lower-bound the cost the optimal algorithm incurs in serving requests to 1-pages. We will denote this cost by $\mathtt{cost}^1_{OPT}(\sigma)$. Consider subphase j in phase i and the preceding subphase. There are a total of $(t_{i,j-1} + n_{i,j})$ distinct 1-pages requested in the two consecutive subphases. Note that if $j = 0$, then the preceding subphase is part of the previous phase, in which case there are a total of $(t_{i-1,p_{i-1}+M} + n_{i,1})$ distinct 1-pages requested in the two consecutive subphases. For notational convenience, we will occasionally denote $t_{i-1,p_{i-1}+M}$ by $t_{i,0}$. Thus, we can always say that the number of pages requested in subphase j of phase i and the preceding subphase is $(t_{i,j-1} + n_{i,j})$. At most $k_{1,i}$ cache slots are used for these requests. Thus, the optimal algorithm spends at least $\max\{0, t_{i,j-1} + n_{i,j} - k_{1,i}\}$ on serving requests to 1-pages in the two consecutive subphases. We can sum up over all pairs of consecutive subphases and get that

$$2\mathtt{cost}^1_{OPT}(\sigma) \geq \sum_{i=1}^{m} \sum_{j=1}^{p_i+M} \max\{0, t_{i,j-1} + n_{i,j} - k_{1,i}\},$$

where $t_{1,0} = 0$. Since the first p_i terms in the inner summation are at least zero, they can be dropped to get:

$$2\texttt{cost}^1_{OPT}(\sigma) \geq \sum_{i=1}^{m} \sum_{j=p_i+1}^{p_i+M} (t_{i,j-1} + n_{i,j} - k_{1,i}).$$

Now we can add the lower bounds for $\texttt{cost}^M_{OPT}(\sigma)$ and $\texttt{cost}^1_{OPT}(\sigma)$ and rearrange as follows:

$$\begin{aligned}
2\texttt{cost}_{OPT}(\sigma) &\geq 2\texttt{cost}^M_{OPT}(\sigma) + 2\texttt{cost}^1_{OPT}(\sigma) \\
&\geq Mn_1 + M\sum_{i=1}^{m-1}(t_i + n_{i+1} - k_{M,i+1}) + \sum_{i=1}^{m}\sum_{j=p_i+1}^{p_i+M}(t_{i,j-1} + n_{i,j} - k_{1,i}) \\
&\geq \sum_{i=1}^{m}\left[Mn_i + \sum_{j=p_i+1}^{p_i+M} n_{i,j}\right] \\
&+ \sum_{i=1}^{m-1}\left[Mt_i + \sum_{j=p_i+1}^{p_i+M} t_{i,j-1}\right] - \sum_{i=2}^{m} Mk_{M,i} - \sum_{i=1}^{m} Mk_{1,i}
\end{aligned}$$

As argued in the two paragraphs preceding Lemma 7, $t_i + t_{i,M+p_i} = k$. Furthermore,

$$t_{i,p_i} \geq t_{i,p_i+1} \geq t_{i,p_i+2} \geq \cdots \geq t_{i,p_i+M}.$$

This means for any $0 \leq j \leq M$, we know that $t_i + t_{i,p_i+j} \geq k$. We can incorporate this into our lower bound for $\texttt{cost}_{OPT}(\sigma)$ as follows:

$$\begin{aligned}
2\texttt{cost}_{OPT}(\sigma) &\geq \sum_{i=1}^{m}\left[Mn_i + \sum_{j=p_i+1}^{p_i+M} n_{i,j}\right] \\
&+ \sum_{i=1}^{m-1}\sum_{j=p_i+1}^{p_i+M}(t_i + t_{i,j-1}) - \sum_{i=2}^{m} Mk_{M,i} - \sum_{i=1}^{m} Mk_{1,i} \\
&\geq \sum_{i=1}^{m}\left[Mn_i + \sum_{j=p_i+1}^{p_i+M} n_{i,j}\right] + \sum_{i=1}^{m-1} kM - \sum_{i=2}^{m} Mk_{M,i} - \sum_{i=1}^{m} Mk_{1,i} \\
&= \sum_{i=1}^{m}\left[Mn_i + \sum_{j=p_i+1}^{p_i+M} n_{i,j}\right] - Mk_{1,1} + \sum_{i=2}^{m} M(k - k_{M,i} - k_{1,i})
\end{aligned}$$

Since there are k slots in the cache, $k_{1,i} \leq k$ and we get that

$$2\texttt{cost}_{OPT}(\sigma) \geq \sum_{i=1}^{m}\left[Mn_i + \sum_{j=p_i+1}^{p_i+M} n_{i,j}\right] - Mk + \sum_{i=2}^{m} M(k - k_{M,i} - k_{1,i}) \quad (1)$$

Now consider two consecutive phases, $i-1$ and i. During this period, $k_{M,i}$ of the optimal algorithm's cache slots ever have M-pages and $k_{1,i}$ of the optimal algorithm's cache slots ever have 1-pages. This means that at least $(k_{M,i} + k_{1,i} - k)$ cache slots have an M-page and a 1-page. If we charge the optimal algorithm $M/2$ for evicting an M-page and $M/2$ for bringing in an M-page, then the optimal algorithm must spend at least $(M/2)(k_{M,i}+k_{1,i}-k)$ in the two consecutive subphases. Adding up over all pairs of consecutive subphases (and multiplying by 2), we get that

$$4\texttt{cost}(\sigma) \geq \sum_{i=2}^{m} M(k_{M,i} + k_{1,i} - k).$$

If we add this to the bound from (1), we have that

$$6\texttt{cost}(\sigma) \geq \sum_{i=1}^{m} \left[Mn_i + \sum_{j=p_i+1}^{p_i+M} n_{i,j} \right] - Mk.$$

■

Proof of Lemma 11. Let σ_i denote the sequence of requests which occur in the i^{th} phase. $\texttt{cost}_{OPT}(\sigma_{i-1}\sigma_i)$ will denote the cost that the optimal algorithm incurs in phases $i-1$ and i. Note that σ_0 will be assumed to be an empty sequence. We will prove that for every i in the range $1 \leq i \leq m$,

$$2\texttt{cost}_{OPT}(\sigma_{i-1}\sigma_i) \geq \sum_{j=1}^{p_i} n_{i,j} \tag{2}$$

This will be sufficient to prove the lemma since

$$4\texttt{cost}_{OPT}(\sigma) \geq \sum_{i=1}^{m} 2\texttt{cost}_{OPT}(\sigma_{i-1}\sigma_i) \geq \sum_{i=1}^{m}\sum_{j=1}^{p_i} n_{i,j}.$$

We will charge the optimal algorithm $(w(p) + w(q))/2$ when it evicts page p on a request to page q. $w(p)/2$ is the cost of evicting page p, and $w(q)/2$ is the cost of bringing in page q. This change in the charging scheme will result in at most a constant additive difference in the cost of serving any sequence. We start by accounting for the cost the optimal algorithm incurs while evicting and bringing in M-pages in phases $i-1$ and i. We denote this cost by $\texttt{cost}^M_{OPT}(\sigma_{i-1}\sigma_i)$. Then we will account for the cost that the optimal algorithm spends evicting and bringing in 1-pages in phases $i-1$ and i, which we will denote by $\texttt{cost}^1_{OPT}(\sigma_{i-1}\sigma_i)$.

Let a be the number of slots that hold the same M-page throughout phases $i-1$ and i. Let b be the number of cache slots that only have 1-pages in the two phases. Let c be the number of cache slots that hold an M-page and some other page in the phase. For each slot i of these c slots, let $P(i)$ be the set of M-pages held in slot i during the phase. Let $c' = |\cup_i P(i)|$. The cost the optimal algorithm incurs is at least $Mc'/2$. We know that $a + b + c = k$ since the three categories partition the cache slots. Also $b + c$ is an upper bound on the number of cache slots that ever hold 1-pages in the two phases, so $b+c \geq k_{1,i}$. In addition $a+c'$ is an upper bound on the number of M-pages that are ever in the cache in the two phases, so $a + c' \geq t_{i-1}$. Putting these inequalities together, we get that

$$\mathtt{cost}^M_{OPT}(\sigma_{i-1}\sigma_i) \geq \frac{Mc'}{2} = \frac{M(a+b+c+c'-k)}{2} \geq \frac{M(k_{1,i}+t_{i-1}-k)}{2}.$$

As argued in the two paragraphs preceding Lemma 7, $k = t_{i-1} + t_{i,0}$. This means we can say that

$$\mathtt{cost}^M_{OPT}(\sigma_{i-1}\sigma_i) \geq \frac{M(k_{1,i}-t_{i,0})}{2} \tag{3}$$

Now we will account for the cost that the optimal algorithm incurs bringing in and evicting 1-pages during the two phases. Consider the j^{th} subphase of phase i and the subphase that immediately precedes it. We know that $t_{i,j-1} + n_{i,j}$ distinct 1-pages are requested in the course of these two subphases. Since the optimal algorithm only uses $k_{1,i}$ cache slots for 1-pages, it must incur a cost of at least $\max\{0, t_{i,j-1} + n_{i,j} - k_{1,i}\}$ in evicting and bringing in 1-pages in these two subphases. Thus, by summing up over all pairs of consecutive subphases, we can say that

$$2 \cdot \mathtt{cost}^1_{OPT}(\sigma_{i-1}\sigma_i) \geq \sum_{j=1}^{p_i} \max\{0, t_{i,j-1} + n_{i,j} - k_{1,i}\}. \tag{4}$$

The factor of 2 comes from the fact that each subphases is counted at most twice.

Consider the first time in the phase when the total number of new 1-pages reaches $M(k_{1,i} - t_{i,0})$. Suppose that this happens in subphase r_i. If $r_i > p_i$, then we can use the bound from (3) to get that

$$\sum_{j=1}^{p_i} n_{i,j} \leq M(k_{1,i} - t_{i,0}) \leq 2 \cdot \mathtt{cost}^M_{OPT}(\sigma_{i-1}\sigma_i),$$

which establishes Inequality 2. Thus, we shall assume that $r_i \leq p_i$. Let $\hat{n}_{i,r_i}$ be such that

$$\sum_{j=1}^{r_i-1} n_{i,j} + \hat{n}_{i,r_i} = M(k_{1,i} - t_{i,0}). \tag{5}$$

We know that $1 \leq \hat{n}_{i,r_i} \leq n_{i,r_i}$. Recall that $t_{i,j}$ is defined to be the number of distinct 1-pages which are requested in subphase j of phase i. This is exactly the number of marked 1-pages which RandCache has in its cache at the end of subphase j of phase i. Since all 1-pages are marked at the end of a subphase, this is also the number of 1-pages which RandCache has in its cache at the end of subphase j ofphase i.

Since RandCache does not evict any 1-pages on requests to M-pages in the first p_i subphases, we know that for $j \leq p_i$, $t_{i,j} - t_{i,0}$ is the number of M-pages which RandCache has evicted to accommodate requests to 1-pages in the first j subphases of phase i. By Lemma 5,

$$t_{i,r_i-1} - t_{i,0} \geq \left\lfloor \frac{\sum_{j=1}^{r_i-1} n_{i,j}}{M} \right\rfloor .$$

Thus, we have that:

$$\begin{aligned} t_{i,r_i-1} + \hat{n}_{i,r_i} &\geq t_{i,0} + \left\lfloor \frac{\sum_{j=1}^{r_i-1} n_{i,j}}{M} \right\rfloor + \hat{n}_{i,r_i} \\ &\geq t_{i,0} + \frac{\sum_{j=1}^{r_i-1} n_{i,j}}{M} + \frac{\hat{n}_{i,r_i}}{M} \\ &= t_{i,0} + \frac{M(k_{1,i} - t_{i,0})}{M} = k_{1,i} \end{aligned} \tag{6}$$

The second inequality comes from the fact that removing the floor can increase the expression by at most an additive $(M-1)/M$ and dividing $\hat{n}_{i,r_i}$ by M will decrease the expression by at least $(M-1)/M$. The series of inequalities gives us that $t_{i,r_i-1} + \hat{n}_{i,r_i} \geq k_{1,i}$.

The total number of new 1-pages is at least $M(k_{1,i} - t_{i,0})$, by the end of phase r_i. This means that by Lemma 5, for any $r_i \leq j < p_i$, at least $k_{1,i} - t_{i,0}$ M-pages have been evicted on requests for 1-pages by the end of subphase j which means that $t_{i,j}$ is at least $t_{i,0} + (k_{1,i} - t_{i,0}) = k_{1,i}$. This fact and (6) are used in the last inequality below. This first inequality is a restatement of (4).

$$\begin{aligned} 2 \cdot \mathtt{cost}^1_{OPT}(\sigma_{i-1}\sigma_i) &\geq \sum_{j=1}^{p_i} \max\{0, t_{i,j-1} + n_{i,j} - k_{1,i}\} \\ &\geq t_{i,r_i-1} + \hat{n}_{i,r_i} + (n_{i,r_i} - \hat{n}_{i,r_i}) - k_{1,i} \\ &\quad + \sum_{j=r_i}^{p_i-1} \max\{0, t_{i,j} + n_{i,j+1} - k_{1,i}\} \\ &\geq n_{i,r_i} - \hat{n}_{i,r_i} + \sum_{j=r_i+1}^{p_i} n_{i,j} \end{aligned} \tag{7}$$

Recall from Inequality (3) and Equation (5) that

$$2 \cdot \texttt{cost}^M_{OPT}(\sigma_{i-1}\sigma_i) \geq M(k_{1,i} - t_{i,0}) = \sum_{j=1}^{r_i - 1} n_{i,j} + \hat{n}_{i,r_i}. \tag{8}$$

Adding the bound for cost for 1-pages (7) to the bound for the cost for M-pages (8), we get that

$$\begin{aligned} 2 \cdot \texttt{cost}_{OPT}(\sigma_{i-1}\sigma_i) &= 2 \cdot \texttt{cost}^M_{OPT}(\sigma_{i-1}\sigma_i) + 2 \cdot \texttt{cost}^1_{OPT}(\sigma_{i-1}\sigma_i) \\ &\geq \sum_{j=1}^{r_i-1} n_{i,j} + \hat{n}_{i,r_i} + (n_{i,r_i} - \hat{n}_{i,r_i}) + \sum_{j=r_i+1}^{p_i} n_{i,j} = \sum_{j=1}^{p_i} n_{i,j} \end{aligned}$$

which is the bound from Inequality 2 that we wanted to establish. ■

Proof of Lemma 8. We will first account for the cost RandCache incurs in bringing 1-pages into the cache. Consider subphase j of phase i. RandCache will incur a cost of 1 on all the new pages of subphase j of phase i. We must now account for the faults which occur on *old* pages (i.e., those page which are requested in subphase j of phase i which were also requested in the previous subphase). Consider the r^{th} request to an old page. This means the $r-1$ old pages are marked. Suppose that g new pages have been requested and that a 1-pages have been evicted on requests to M-pages at this point. There are $t_{i,j-1} - (r-1)$ pages which were requested in the last subphase which have not been requested yet in the current subphase. Call this set of pages S. There are $t_{i,j} - (r-1) - g - a$ random pages from S in the cache. This means that there are $a + g$ pages in S which are not in the cache. The probability that the page requested in the r^{th} request to an old page is not in the cache is $(g+a)/(t_{i,j-1} - (r-1))$. If we let $a_{i,j}$ denote the total number of 1-pages which are evicted on requests to M-pages in subphase j of phase i, then the r^{th} request to an old page is a fault with probability at most $(n_{j,i} + a_{i,j})/(t_{i,j-1} - r + 1)$.

Thus, the total expected cost incurred in serving 1-pages in subphase j of phase i is at most

$$n_{i,j} + \sum_{r=1}^{t_{i,j-1}} \frac{n_{i,j} + a_{i,j}}{t_{i,j-1} - r + 1} \leq n_{i,j} + (n_{i,j} + a_{i,j}) \log(t_{i,j-1}) \leq n_{i,j} + (n_{i,j} + a_{i,j}) \log k.$$

This first term comes from the fact that RandCache always incurs a cost for serving the new pages. The sum comes from the expected cost in serving old pages. Thus, the total expected cost of serving 1-pages over the entire sequence is at most

$$(\log k + 1) \sum_{i=1}^{m} \sum_{j=1}^{p_i+M} n_{i,j} + \log k \sum_{i=1}^{m} \sum_{j=1}^{p_i+M} a_{i,j}.$$

Let b_i be the number of slots which are occupied by an M-page and then later by a 1-page in phase i. Using the same reasoning, we can upper bound the cost of the algorithm in serving M-pages by

$$(\log k + 1) \sum_{i=1}^{m} M n_i + \log k \sum_{i=1}^{m} M b_i.$$

Putting these bounds together, we get that

$$\mathtt{cost}_{RC}(\sigma) \leq (\log k + 1) \left[\sum_{i=1}^{m} M n_i + \sum_{j=1}^{p_i+M} n_{i,j} \right] + \log k \sum_{i=1}^{m} \left[M b_i + \sum_{j=1}^{p_i+M} a_{i,j} \right].$$

The number of times a 1-page is evicted on a request to an M-page can be at most the number of times a M-page is evicted on a request to a 1-page plus k. Thus,

$$\sum_{i=1}^{m} \sum_{j=1}^{p_i+M} a_{i,j} \leq \sum_{i=1}^{m} b_i + k.$$

Which gives us that

$$\mathtt{cost}_{RC}(\sigma) \leq (\log k+1) \left[\sum_{i=1}^{m} M n_i + \sum_{j=1}^{p_i+M} n_{i,j} \right] + k \log k + \log k \left[\sum_{i=1}^{m} (M+1) b_i \right].$$

By Lemma 6, for each i,

$$M b_i \leq \max \left\{ M, \sum_{j=1}^{p_i+M} n_{i,j} \right\} \leq M + \sum_{j=1}^{p_i+M} n_{i,j}.$$

Putting it all together, we get that

$$\begin{aligned} \mathtt{cost}_{RC}(\sigma) &\leq \left(\left(2 + \frac{1}{M} \right) \log k + 1 \right) \left[\sum_{i=1}^{m} M(n_i + 1) + \sum_{j=1}^{p_i+M} n_{i,j} \right] + k \log k \\ &\leq 2 \left(\left(2 + \frac{1}{M} \right) \log k + 1 \right) \sum_{i=1}^{m} \max \left\{ M, \left[M n_i + \sum_{j=1}^{p_i+M} n_{i,j} \right] \right\} + k \log k \end{aligned}$$

■

References

[ACN00] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1-2):203–218, 2000.

[BBBT97] Yair Bartal, Avrim Blum, Carl Burch, and Andrew Tomkins. A polylog(n)-competitive algorithm for metrical task systems. In *Proc. 29th Symp. Theory of Computing*, pages 711–719, 1997.

[BCL98] Yair Bartal, Marek Chrobak, and Lawrence L. Larmore. A randomized algorithm for two servers on the line. In *Proc. 6th European Symp. on Algorithms*, Lecture Notes in Computer Science, pages 247–258. Springer, 1998.

[CKPV91] Marek Chrobak, Howard Karloff, Tom H. Payne, and Sundar Vishwanathan. New results on server problems. *SIAM Journal on Discrete Mathematics*, 4:172–181, 1991.

[Fia01] Amos Fiat. Personal communication, 2001.

[FKL$^+$91] Amos Fiat, Richard Karp, Michael Luby, Lyle A. McGeoch, Daniel Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[MMS90] Mark Manasse, Lyle A. McGeoch, and Daniel Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[MS91] Lyle McGeoch and Daniel Sleator. A strongly competitive randomized paging algorithm. *Journal of Algorithms*, 6:816–825, 1991.

[ST85] Daniel Sleator and Robert E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[You94] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11:525–541, 1994.

```
Algorithm RandCache(p)
Let $p$ denote the requested page.
(1) if $p$ is marked,
        do nothing and return.
(2) if $p$ is in the cache and unmarked,
        mark the page and return.
(3) if $w(p) = M$,
(4)     if there are unmarked $M$-pages in the cache,
(5)         Pick one at random and evict it.
            Mark page $p$ and bring it into the cache.
(6)     else if there are unmarked 1-pages in the cache,
(7)         Evict a random unmarked 1-page.
            Mark page $p$ and bring it into the cache.
(8)     else
(9)         if no $M$-pages have been evicted in the subphase
            to accommodate a request for a 1-page,
(10)            $N_2 \leftarrow N_2 + 1$.
(11)        if $N_2 < M$
(12)            Call NewSubPhase(p)
(13)        else Call NewPhase(p)
(14) if $w(p) = 1$,
(15)    if there are unmarked 1-pages in the cache,
(16)        Evict a random unmarked 1-page.
            Mark page $p$ and bring it into the cache.
(17)        if $p$ was not requested in the previous subphase
(18)            $N_1 \leftarrow N_1 + 1$
(19)    else if there are unmarked $M$-pages in the cache,
(20)        if $N_1 \geq M$ or there are no 1-pages in the cache,
(21)            Evict a random unmarked $M$-page,
                Mark page $p$ and bring it into the cache.
(22)            if $p$ was not requested in the previous subphase
(23)                $N_1 \leftarrow N_1 + 1$
(24)            $N_1 \leftarrow N_1 - M$.
(25)        else Call NewSubPhase(p)
(26)    else
(27)        if no $M$-pages have been evicted in the subphase
            to accommodate a request for a 1-page,
(28)            $N_2 \leftarrow N_2 + 1$.
(29)        if $N_2 < M$
(30)            Call NewSubPhase(p)
(31)        else Call NewPhase(p)
end
```

Figure 1: Randomized Algorithm for Weighted Caching with two weights.

```
Procedure NewPhase(p)
    Unmark all $M$-pages.
    $N_1 \leftarrow 0$.
    $N_2 \leftarrow 0$.
    Call NewSubPhase(p).
```

Figure 2: Start a new phase.

```
Procedure NewSubPhase(p)
Unmark all 1-pages.
Call RandCache(p).
```

Figure 3: Start a new subphase.