Page Replacement with Multi-Size Pages and Applications to Web Caching

Sandy Irani *
Information and Computer Science Department
University of California, Irvine
Irvine, CA, 92697
irani@ics.uci.edu

Abstract

We consider the paging problem where the pages have varying size. This problem has applications to page replacement policies for caches containing World Wide Web documents. We consider two models for the cost of an algorithm on a request sequence. In the first, (the FAULT model) the goal is to minimize the number of page faults. In the second, (the BIT model) the goal is to minimize the total number of bits that have to be read into the cache. We show offline algorithms for both cost models that obtain approximation factors of $O(\log k)$, where k is the ratio of the size of the cache to the size of the smallest page. We show randomized online algorithms for both cost models that are $O(\log^2 k)$-competitive. In addition, if the input sequence is generated by a known distribution, we show an algorithm for the FAULT model whose expected cost is within a factor of $O(\log k)$ of any other online algorithm.

1 Introduction

The problem of developing and evaluating page replacement policies for a two-level store of memory has been, and continues to be, a fundamental problem in computing systems. With the advent of the World Wide Web, these issues have resurfaced with some significant variations. Developing Web cache management policies is an extremely important problem today. The demand for resources on the Web will only increase as its enormous popularity continues to grow and as typical uses of the Web become more sophisticated. Maintaining a cache of pages at various

*Supported in part by NSF Grant CCR-9309456 and CCR-9625844.

points can dramatically reduce demand on the network as well as latency seen by the user. A cache can be maintained at a Web client (e.g., caches built into Web browsers) [BCC+95] or by a Web server [KMR95]. Caches can also be very effective when placed in the network itself [DHS93].

There are several possible measurements that can be used to study the effectiveness of a caching policy. One may wish to minimize the number of requests reaching popular servers. Viewed another way, this amounts to minimizing the number of times a client must wait for a document to be retrieved from a remote cite. We call this model the FAULT model in which the number of cache faults are counted. Alternatively, one may wish to minimize the total volume of traffic resulting from document requests. In this model, which we call the BIT model, the cost is the total size of the pages that are requested on cache faults. Alternatively, one may want to minimize the latency experienced by the user. Measuring latency requires the development of an accurate model for predicting the time to fetch a requested document from the appropriate server. We do not consider this latter model here. The FAULT and BIT models, which we do consider here, are the most common in the literature on Web caching [WAS+96, AW96, M96].

Empirical studies have indicated that the choice of page replacement policy for Web caches can have a profound effect on the utility of the cache [WAS+96, M96]. Conventional wisdom that has been gained in the context of page replacement for CPU caches does not necessarily transfer to Web caches. In particular, the Least-Recently-Used policy, which on a fault evicts the page whose next request is farthest in the future, usually performs quite well for CPU caches but can be highly inferior to other policies for Web caches. One reason is that LRU does not take into account the size of the documents in choosing a page to evict. This is not a problem in traditional CPU caches where data is divided into uniform size blocks. However, Web documents can vary dramatically in size depending largely on the type of information they contain (video, audio, text, etc). Developing effective page replacement policies for documents that vary in size is the main focus of this paper.

We present here a theoretical analysis of page replacement policies for multi-size pages using measures of analysis that are now standard in the literature of online algorithms. When evaluating an algorithm that knows nothing about future requests (an *online algorithm*), we use competitive analysis. Let $\texttt{cost}_A(\sigma)$ be the cost incurred by an online algorithm A on the input sequence σ, be it under the BIT or the FAULT model. Let OPT be the optimal offline algorithm, and let $\texttt{cost}_{OPT}(\sigma)$ be the cost incurred by the optimal offline algorithm on input σ. We say that the online algorithm A is c-competitive if there exists a constant b such that on every request sequence σ,

$$\texttt{cost}_A(\sigma) \leq c \cdot \texttt{cost}_{OPT}(\sigma) + b.$$

The *competitive ratio* of the algorithm A, denoted c_A is the infimum over c such that A is c-competitive.

We use a slightly different version of the paging problem that does not change the analysis significantly, although it does better suit the application of Web caching: when a page is requested, the algorithm is not forced to bring the page into the cache. If the page is not in the cache, the algorithm can pay the price of a fault and leave the page outside the cache. In this case, it would have to pay again the next time the page is requested. To avoid confusion with the standard version where the requested page must be brought into the cache, we call our version of the problem the MULTI-SIZE OPTIONAL PAGING PROBLEM.

One issue that we do not address is that of cache consistency. We assume that the pages are static. Thus, if a page is referenced and a copy of the page resides in the cache, the request can be immediately satisfied. One way to deal with out-of-date pages is to treat requests to pages that have not been retrieved within a certain time limit as requests to new pages. The details of such a scheme are beyond the scope of this paper.

The best competitive ratio that can be achieved by any deterministic online algorithm is achieved by the familiar *Least-Recently-Used* (LRU):

Theorem 1 *[FKIP96] LRU is $(k+1)$-competitive for the* MULTI-SIZE OPTIONAL PAGING PROBLEM *in both the* FAULT *and the* BIT *models. This bound is the best achievable by a deterministic online algorithm in both cost models.*

Throughout this paper, k denotes the maximum number of pages that can fit into the cache (i.e. the size of the cache divided by the size of the smallest page). The proof of Theorem 1 involves straightforward adaptations of the bounds for uniform size pages by Sleator and Tarjan ([ST85]). Since LRU has been observed to be sub-optimal in practice, these results indicate that a more refined analysis is needed to evaluate Web cache replacement policies.

Although the problem we wish to solve is inherently online, often algorithms that work with partial information attempt to imitate the behavior of the optimal offline algorithm to the extent possible with their limited information. For this reason, it is useful to have an understanding of the behavior of the optimal offline algorithm. Indeed, the randomized online algorithm and probabilistic analysis discussed below depend heavily on the offline approximation algorithm presented in Section 4. When the pages all have the same size, there is a simple rule, due to Belady, for obtaining the optimal replacement policy [Bel66]: on a fault evict the page whose next request is farthest in the future. Such a simple rule for the multi-size case does not seem to exist. In fact the problem of devising the optimal replacement policy for a given sequence of requests is NP-hard in the BIT model [Fiat97]. It is unknown whether the problem is in P for the FAULT model. We develop a approximation algorithms for the FAULT and BIT model and prove the following theorem:

Theorem 2 *There is a polynomial-time offline algorithm for the* FAULT *model and a polynomial-time offline algorithm for the* BIT *model whose cost is guaranteed to be within $O(\log k)$ of the optimal offline algorithm for* MULTI-SIZE OPTIONAL PAGING PROBLEM .

The algorithm uses Belady's rule to pick among pages of similar size. The factor of $O(\log k)$ comes from balancing among pages of different sizes. Fortunately, the rule for picking among pages of different sizes requires no information about the future which means that it can be combined with known online algorithms for page replacement with pages of uniform size. This is the idea involved in the online randomized algorithms in Section 5. We use a variation of the Randomized Marking Algorithm developed in [FKL+91] combined with the offline approximation presented here. Although we borrow the basic structure of the algorithm from [FKL+91], it was necessary to develop an alternate proof of their bound based on a potential function in order to combine it with the proof for the offline approximation algorithm. The alternate proof is implicit in the proof of the following theorem.

Theorem 3 *There is a randomized online algorithm for the* BIT *model and a randomized online algorithm for the* FAULT *model, both of which achieve a competitive ratio of $O(\log^2 k)$ for the* MULTI-SIZE OPTIONAL PAGING PROBLEM .

Finally we turn to probabilistic analysis. Again, we combine the approximation algorithms in Section 4 with a known algorithm for the uniform size case. We use the algorithm of Lund, Phillips and Reingold for paging when the sequence is generated by an arbitrary known distribution [LPR94]. The cost of the Lund *et al.* algorithm is within a factor of 4 of any online algorithm that knows the distribution in advance. We prove the following theorem:

Theorem 4 *Given a distribution over input sequences, there is a randomized algorithm for the* FAULT *model whose expected cost is within a factor of $O(\log k)$ of the expected cost of any online algorithm that knows the distribution generating the input. The expectation is taken over the random choices of the algorithm as well as the random choices made in generating the sequence.*

The proof of Theorem 4 follows the structure of the proof in [LPR94] quite closely. However, some extra machinery is required.

2 Related Work

To our knowledge, this is the first paper that gives a theoretical analysis of paging with multi-size pages. Since these results were obtained, there have been several papers that address paging with multi-size pages where the cost of bringing a page into the cache is arbitrary. Cao and Irani [CI97] and Young [Young98] analyze a generalization of an algorithm called *greedy-dual* developed by Young [Young94]. This deterministic algorithm is shown to be $(k+1)$-competitive. Young shows that it is in fact $(k+1/k-h+1)$-competitive when the optimal algorithm has a cache size of h. Cao and Irani also perform an empirical evaluation of the algorithm using trace data. Cohen and Kaplan then give an alternate proof of the $(k+1/k-h+1)$-competitiveness of this algorithm using a linear programming formulation of the problem [CK99]. They use the linear programming formulation to give an offline algorithm that gives an approximate solution when the size of the largest page is small with respect to the size of the cache.

Albers, Arora and Khanna [AAK99] examine the general cost model and obtain an $O(1)$-approximation algorithm for the offline problem that uses an additional amount of memory that is $O(1)$ times the size of the largest page. When no additional memory is allowed, they give an algorithm that achieves an $O(\log(M+C))$ approximation factor, where M is the size of the cache and C is the largest cost to bring a page into the cache. Finally, they give a randomized algorithm for the BIT model that achieves a competitive ratio of $O(\ln(1+1/c))$ while using $M(1+c)$ memory for any $c > S/M$, where S is the size of the largest page.

3 Preliminaries

In Web terminology, the objects being cached are called *documents*. A Web page can be comprised of many documents. In order to stay consistent with the terminology of the online algorithms literature, we refer to the objects being cached as *pages*.

The size of the pages are measured by the number of bits they contain. In the discussion that follows, we normalize all page sizes so that the smallest page has size 1. This may mean that a page may contain a fractional number of bits, but it does not effect the algorithms or analysis. The size of a page p is denoted by $|p|$. In some cases, we will define our algorithms in terms of a relaxed algorithm that is allowed to partially evict a page. In these cases, $|p|_{out}$ (resp. $|p|_{in}$) will denote the number of bits of p that the algorithm has outside (resp. inside) the cache.

In all of the algorithms, the pages are divided into at most $\lfloor \log k \rfloor + 1$ classes. Class l contains pages of size at least 2^l and less than 2^{l+1}. We refer to the pages in class l as *l-pages*. We say that the class size of x (denoted $cs(x)$) is the class that contains pages of size x (i.e. $\lfloor \log x \rfloor$).

In all the proofs in this paper, we compare the cost of an approximation or online algorithm with the cost of the optimal offline algorithm. This is achieved by examining an arbitrary sequence of page requests and analyzing the cost of both algorithms as they service the requests in that sequence. Since we are working with the version of paging where we are not obliged to bring the requested page into the cache, we can view a request as occurring in the following three steps:

1. If an algorithm does not have the requested page in the cache, it brings in the requested page (and may afterwards exceed its memory capacity).

2. The request is served and the time of the next request to that page is updated.

3. The algorithm evicts at least enough pages so that it does not exceed its memory capacity.

In every proof, we charge an algorithm for the pages it evicts instead of the pages it brings in to the cache. Bringing a page into the cache is free. Changing the charging scheme in this manner changes the cost of a given algorithm on any particular sequence by at most an additive factor of k. We will use e to denote the amount by which the cache is exceeded. For any set of pages S, the *most distant* page in S is the page in S whose next request is farthest in the future. Let $S_A(l)$ be the set of l-pages that algorithm A has in the cache. In general, we use lower case letters to denote the size of a set, so $s_A(l)$ is the number of pages from class l that A has in the cache. We use $b_A(l)$ to refer to the sum of the sizes of the l-pages that A has in the cache.

4 Offline Algorithms

4.1 The Fault Model

Figure 1 is a description of the algorithm that we call the Offline Fault Model Algorithm (OFMA). The essential idea of the algorithm is that we know how to make decisions in choosing which page

```
OFFLINEFAULTMODELALGORITHM:
Consider a request to l-page p:
(1) if p is in the cache, do nothing.
(2) else bring p into the cache.
(3) if the size of the cache is exceeded,
(4)     for all j, do twice:
(5)         if there is a j-page in cache,
(6)             evict the most distant
                j-page in the cache.
```

Figure 1: Offline Approximation Algorithm for the Fault Model.

to evict within a class since the page sizes within a class only vary by a factor of at most two: we can use Belady's rule to evict the two most distant pages. However, when we must make room in the cache, we may not know from which class to pick. The algorithm solves this problem by evicting two pages from every class. Since the algorithm pays for evictions, we get a factor of $2 \log k$. This is established more formally in the following theorem:

Theorem 5 *The number of faults incurred by algorithm OFMA on any sequence of requests is within a factor of* $2 \log k$ *of the number of faults incurred by the optimal offline algorithm.*

Proof. For the purposes of our proof, we give the optimal offline algorithm (OPT) a special discount on evicting pages. On requests in which OFMA evicts a page, we allow OPT for the price of one eviction, to evict from each class either the two pages that OFMA has chosen to evict or one page of its choice. Any additional evictions made by OPT are undiscounted. Certainly the optimal algorithm with this special discount performs at least as well as the optimal algorithm without the discount. Furthermore, OPT will not perform any undiscounted evictions unless it is forced to make room in its cache. We claim that the behavior of an optimal algorithm with the discount is exactly the behavior of OFMA. Since the optimal gets at most $2 \log k$ evictions for the price of one, if the claim is true, our cost is at most $2 \log k$ times the optimal.

We prove our claim by induction on t. Suppose that up to time t the behavior of OPT and OFMA are the same. The next request at time $t + 1$ is already in OPT's cache if and only if it is in the OFMA's cache. If the page is not in either's cache, then it is brought in. The capacity of OPT's cache is exceeded if and only if the capacity of OFMA's cache is exceeded. If the capacity is not exceeded, then we go on to the next request.

Suppose that the capacity of the cache is exceeded. OPT must evict at least one page, incurring a cost of one. Since OPT must evict at least one page, it may as well take advantage of the special discount and evict one or two pages from each class (depending on whether it chooses to evict the same pages as OFMA). Remember that bringing pages into the cache is free but evicting pages has a cost.

The remaining fact to prove is that if for each class j such that the algorithms have at least two j-pages in the cache, OPT always chooses to evict the same j-pages as OFMA. Suppose the two most distant j-pages are p and q. These are the two pages that OFMA evicts. If OPT does not evict p and q, then it evicts some other page r.

We can convert OPT into another algorithm OPT' that evicts p and q at time t and does at least as well as OPT. If OPT evicts either p or q at some time t' before the next request to r, then OPT' evicts r at time t'. The rest of the eviction choices remain exactly the same except for the fact that OPT may have to evict p or q where OPT' has already evicted them. Note that until the next request to r, OPT' has at least as much unused space in the cache as OPT. This is because the total space taken by p and q is at least the space taken up by r. After the request to r, the only difference in their configurations is that until the next requests to p and q, OPT' does not have those pages in the cache while OPT may have p or q in memory. Since we do not penalize an algorithm for bringing a page into memory, OPT' is at least as well off as OPT. $\square$

4.2 The Bit Model

Figure 2 contains a description of the algorithm that we call the *Offline Bit Model Algorithm* (OBMA). A counter is maintained for each class. $c(l)$ is the current value of the counter for class l. All counters are initialized to 0 at the beginning of the sequence. The idea behind this algorithm is similar to that of OFMA. The problem here, however, is that we cannot afford to evict a large page if the memory capacity is only exceeded by a small amount. What we would like to do instead is to evict e bits from every class if the memory capacity is exceeded by e bits. Since we cannot evict partial pages, if e is smaller than the pages in a class, we just add to that class's counter. When the counter is large enough to account for the cost of evicting a page from that class, we cash in the counter and evict a page.

```
OFFLINEBITMODELALGORITHM:
Consider a request to a page $p$ of size $x$
from class $l$:
(1) Bring $p$ into the cache if not there.
(2) if the capacity of cache is exceeded by $e > 0$ bits.
(3)       if $\sum_{j \le cs(e)} b_{OBMA}(j) \le e$
(4)            let $i$ be the smallest number greater than $cs(e)$ such
               thet there is at least one $i$-page in the cache.  Let $m$
               be the size of the most distant $i$-page.
(5)       else $m \leftarrow e$
(6)       for all $j \le cs(e)$:
(7)            Continue until the total size of the pages evicted is at
               least $m$ or there are no $j$-pages in the cache:
(8)                 Evict the most distant $j$-page.
(9)       for all $j > cs(e)$:
(10)           if there is a $j$-page in cache,
(10)                $c(j) \leftarrow c(j) + m$;
(11)                while $c(j) \ge$ size of most distant $j$-page $q$,
(12)                     Evict $q$.
(13)                     $c(j) \leftarrow c(j) - |q|$.
```

Figure 2: Offline Approximation Algorithm for the Bit Model.

The analysis makes use of a potential function. (For an introduction to the use of potential functions see [T85]). Before defining the potential function, some preliminary definitions will be necessary. Define $S_{OBMA}(l,t)$ to be the set of pages in class l that $OBMA$ currently has in the cache whose next request occurs on or after time t. A page that is never requested again is assumed to be requested at time ∞. $S_{OPT}(l,t)$ is defined similarly for the optimal algorithm OPT. Let $b_{OBMA}(l,t)$ denote the sum of the sizes of pages in the set $S_{OBMA}(l,t)$. $b_{OPT}(l,t)$ is defined similarly for OPT. Let t be the time of the current request. Define

$$\alpha_l = \max_{t' \ge t}\{b_{OBMA}(l,t') - b_{OPT}(l,t')\}.$$

$$\phi_l = \begin{cases} 0 & \text{if } \alpha_l \le 0 \\ \max\{\alpha_l, 2^{l+2}/5\} - c(l)/5 & \text{if } \alpha_l > 0 \end{cases}$$

The value of the potential function is just the sum of the ϕ_l's:

$$\Phi = \sum_l \phi_l.$$

Notice that after each request is processed, $c(l) \leq 2^{l+1}$ which means that ϕ_l is always non-negative.

We also use another potential function Λ that is just the sum of all the counters of OBMA:

$$\sum_l c(l) = \Lambda.$$

We use the notation $\Delta\Phi$ to denote the change in a potential function Φ over some given event. That is, it will denote the value of the potential function after the event minus the value of the potential function before the event. We charge each algorithm for the pages they evict and prove the following lemma:

Lemma 6 *After each stage of the request:*

$$\textit{OBMA's cost } + \Delta\Lambda \leq 5(\log k + 4)(\textit{OPT's cost} - \Delta\Phi).$$

The lemma is sufficient to establish that OBMA is $5(\log k + 4)$-competitive since both potential functions are always non-negative and both start at 0.

Before proving Lemma 6, we prove the following small lemma:

Lemma 7 *Suppose that OBMA evicts the most distant l-page p and after the eviction, $\alpha_l > 0$. Then α_l has decreased by $|p|$.*

Proof. Let $\hat{t}$ be the time that p is requested next. After p is evicted, all the pages that OBMA has in the cache are requested before time $\hat{t}$. Thus, after p is evicted, $b_{OBMA}(l,t) = 0$ for all $t \geq \hat{t}$. Since $\alpha_l > 0$ after p is evicted, it must be the case that $\max_{t' \geq t}\{b_{OBMA}(l,t') - b_{OPT}(l,t')\}$ is maximized when $t' < \hat{t}$. Suppose it is maximized for some $\bar{t} = t'$. Since $\bar{t} < \hat{t}$, $p \in S_{OBMA}(l,\bar{t})$ before p is evicted but not after it is evicted. Thus, $b_{OBMA}(l,\bar{t})$ decreases by $|p|$ when p is evicted. This means that $\max_{t' \geq t}\{b_{OBMA}(l,t') - b_{OPT}(l,t')\}$ decreases by at least $|p|$ when p is evicted. □

Proof of Lemma 6. Consider a request to page p at time t. Let $l = cs(p)$. We break the analysis into four cases:

Case 1: Both OPT and OBMA have p in the cache. Neither algorithm incurs any cost. Since both algorithms have p in the cache, the page p is in $S_{OBMA}(t',l)$ if and only if it is in $S_{OPT}(t'.l)$ for all t'. This is true before the request to p at time t and after the next request to p is updated to some later time. Thus, Φ does not change.

Case 2: OBMA does not have p in the cache, and OPT does have p in the cache. Consider the moment before OBMA brings p into the cache. Since OPT already has p in the cache, $b_{OPT}(t,l) = b_{OPT}(t+1,l) + |p|$. Since OBMA does not have p in the cache, $b_{OBMA}(t,l) = b_{OBMA}(t+1,l)$. Thus,

$$\begin{aligned}
& b_{OBMA}(l,t) - b_{OPT}(l,t) \\
= \ & b_{OBMA}(l,t+1) - (b_{OPT}(l,t+1) + |p|) \\
\leq \ & \max_{t' \geq t}\{b_{OBMA}(l,t') - b_{OPT}(l,t')\} - |p|.
\end{aligned}$$

This means that before the page is brought in, $b_{OBMA}(l,t) - b_{OPT}(l,t)$ is $|p|$ less than $\alpha_l = \max_{t' \geq t}\{b_{OBMA}(l,t') - b_{OPT}(l,t')\}$. Thus, after the page is brought in, $b_{OBMA}(l,t)$ increases by $|p|$, but α_l remains unchanged.

Then the time of the next request to p changes to some time $\hat{t}$ in the future. Since OPT and OBMA both have p in the cache, Φ does not change. (This is just the same as the argument in Case 1).

Now suppose that OBMA has exceeded its memory capacity by bringing in page p. It evicts some pages and adds to the counter of other classes. How much does OBMA pay? For $j \leq cs(e)$, the sum of the sizes of the pages that OBMA evicts is at most $m + 2^{j+1}$. For $j > cs(e)$, m points are added to the counter of m. This amounts to a total amortized cost of:

$$\begin{aligned}(m \log k) + \sum_{i=0}^{cs(e)} 2^{i+1} &\leq (m \log k) + 4 \cdot 2^{cs(m)} \\ &\leq (m \log k) + 4m \\ &\leq (\log k + 4)m.\end{aligned}$$

If the algorithm trades in s points from $c(l)$ in exchange for evicting a page of size s, its amortized cost (cost + $\Delta\Lambda$) is 0.

We must now prove that Φ decreases by at least $m/5$.

Let $e(j) = b_{OBMA}(j) - b_{OPT}(j)$. OBMA has $e(j)$ more bits belonging to j-pages in memory than OPT. We make the following claim:

Claim 8 *For $j \leq cs(e)$, if $e(j) > 0$, then ϕ_j decreases by at least $e(j)/5$. For $j > cs(e)$, if $e(j) > 0$, then ϕ_j decreases by at least $m/5$.*

Note that in either case, as long as $e(j) > 0$, then ϕ_j decreases by at least $e(j)/5$ since $m \geq e \geq e(j)$. The claim will be sufficient to establish that Φ decreases by at least $m/5$ because if m was chosen to be e, then we have that

$$\sum_{e(j)>0} e(j) \geq \sum_j e(j) = e = m,$$

and as long as ϕ_j decreases by $e(j)/5$ for each class with $e(j) > 0$, the total decrease is at least $m/5$. Note that ϕ_j does not increase for any j when OBMA evicts a page, so we need not be concerned with those classes j for which $e(j) \leq 0$.

Alternatively, suppose m was chosen to be the smallest page in the cache from a class larger than $cs(e)$. We know that in this case the sum of the sizes of the pages in classes of size $cs(e)$ or less are not enough to compensate for the excess that OBMA has in the cache. Thus, for some $j > cs(e)$, the sum of the sizes of the j-pages that OBMA has in cache is more than OPT. Thus, we are guaranteed that $e(j) > 0$ for some $j > cs(e)$. Then, by the claim, ϕ_j decreases by at least $m/5$.

Now we must prove the claim. We break the analysis into two cases:

$j > cs(e)$: First $c(j)$ is incremented by m. Note that if the algorithm has at least one page from class j in the cache, then $j \geq cs(m)$. This implies that $m < 2^{j+1}$. Since $e(j) > 0$, we know that $\alpha_j > 0$. This is because $\max_{t' \geq t}\{b_{OBMA}(j,t') - b_{OPT}(j,t')\} \geq e(j)$ since t' can always be chosen to be t. Before $c(j)$ is incremented by m, $c(j) < 2^{j+1}$ which means that $\phi_j \geq 2^{j+1}/5$. Since $m < 2^{j+1}$, when $c(j)$ is incremented by m, ϕ_j will decrease by at least $m/5$.

Now we must establish that if a page from class j is evicted and $c(j)$ is decremented, then ϕ_j does not increase. Suppose the evicted page has size s. If after the eviction, α_j becomes 0, then ϕ_j also becomes 0. Since ϕ_j is always non-negative, it certainly has not increased. Now suppose that $\alpha_j > 0$ after the eviction. By Lemma 7, α_j decreases by at least s. This means that $\max\{\alpha_j, 2^{j+2}/5\}$ decreases by at least $s - 2^{j+2}/5$ and ϕ_j decreases by at least $(s - 2^{j+2}/5) - s/5$. Since $s \geq 2^j$, ϕ_j does not increase.

$j \leq cs(e)$: suppose that $\phi_j > 0$ after the evictions. Let s be the number of bits from class j that OBMA evicts. We know that $s \geq e(j)$. We also know that $s \geq 2^j$ since all j-pages have at least 2^j bits. By Lemma 7, α_j decreases by at least s bits. Thus, ϕ_j decreases by at least

$$\max\{2^j, e(j)\} - \frac{2^{j+2}}{5} > \frac{e(j)}{5}.$$

Now suppose that $\phi_j = 0$ after the evictions. We know that before the evictions $\alpha_j \geq e(j) > 0$ because $\alpha_j = \max_{t' \geq t}\{b_{OBMA}(j,t') - b_{OPT}(j,t')\}$ and $e(j) = b_{OBMA}(j,t) - b_{OPT}(j,t)$. Thus, ϕ_j decreases by at least

$$\max\left\{\alpha_j, \frac{2^{j+2}}{5}\right\} - \frac{2^{j+1}}{5} \geq \frac{\alpha_j}{2} \geq \frac{e(j)}{5}.$$

Case 3: Both OBMA and OPT fault. Both algorithms bring p into the cache in step 1. The argument that Φ does not change is the same as Case 1. Similarly, when the time of the next reference to p is updated, Φ also does not change. Now OPT may have to evict some pages in order to keep the number of bits in the cache at most k. Suppose OPT evicts h bits from class j. OPT pays h. α_j (and hence ϕ_j) increases by at most h. If the capacity of the cache is exceeded, OBMA may have to evict some pages. This part of the argument is identical to Case 2.

Case 4: OPT faults and OBMA does not fault. OPT brings p into the cache. Φ can only decrease. Then OPT may evict some pages in order not to exceed the the cache capacity of k. If OPT evicts h bits, then it pays h and Φ can increase by at most h. □

5 Randomized Online Algorithms

The randomized algorithms for both the fault and the bit model will be described in terms of a relaxed version that will be allowed to evict a portion of a page. The real algorithms will evict the entire page as soon as any portion of a given page have been evicted. This guarantees that the sizes of the pages in the cache never exceed the capacity of the cache. We will also charge the relaxed versions the cost of evicting the entire page (a cost of 1 in the Fault model and the size of the page in the BIT model), as soon as any of the bits of a given page are evicted. This ensures that the cost of the real algorithm and its relaxed version are the same.

Each of the randomized algorithms makes use of a marking scheme as introduced in [KMRS88]. Figure 3 shows the eviction routine that is common to both algorithms. It is called whenever it is necessary to evict C bits belonging to pages of class l. Both algorithms work according to the following principles. If there is a request to a page p, it is brought into the cache and marked. Only unmarked pages are ever evicted from the cache. If it is necessary to evict an l-page and all the l-pages in the cache are marked, then the algorithm unmarks all the l-pages. For each class, the sequence is divided into phases. The current phase for class l (also called an l-phase) ends and a new one begins whenever the pages of that class are unmarked as in step (8) of the eviction algorithm. The set of marked l-pages is always exactly the set of l-pages that have been requested in the current l-phase. At the end of a phase, the set of l-pages in the cache are exactly those l-pages that have been requested in the l-phase.

Throughout Sections 5.1 and 5.2, we will denote the size of a page by $|p|$. $|p|_{in}$ (resp. $|p|_{out}$) will denote the number of bits of page p that RFMA has inside (resp. outside) the cache.

EVICT(l, C):
(1) **if** there is a page q that is partially evicted:
(2) Let $x \leftarrow \min\{|q|_{in}, C\}$.
(3) Evict x bits of q.
(4) $C \leftarrow C - x$.
(5) **while** $C \neq 0$:
(6) **if** there are l-pages in the cache:
(7) **if** there are no unmarked l-pages:
(8) Unmark all l-pages.
(9) Pick an unmarked l-page q at random.
(10) Let $x \leftarrow \min\{|q|, C\}$.
(11) Evict x bits of q from the cache.
(12) $C \leftarrow C - x$.
(13) **if** there are fewer than 2^l bits belonging to unmarked l-pages in the cache, evict them.

Figure 3: Eviction Algorithm used in both the randomized algorithms for the Fault and Bit Models.

5.1 The Fault Model.

We now describe the Randomized Fault Model Algorithm (RFMA). As with the algorithm in the previous section, RFMA resolves the problem of which class to evict from by evicting from every class. The decision of which page to evict from a given class is made according to the Randomized Marking Algorithm of [FKL+91]. Although the idea appears simple, in order to combine the proofs of the two algorithms, it is necessary to prove the competitiveness of the Randomized Marking Algorithm using a potential function argument that is substantially different than the original proof.

```
RandomizedFaultModelAlgorithm:
Consider a request to l-page p:
(1) Bring p into the cache if not there.
(2) Mark page p.
(3) if p was requested in the previous phase,
(4)     Let u_l be the number of bits belonging to
        unmarked l-pages in the cache.
(5)     Evict(l, min{u_l, |p|_out}).
(6) if total sizes of pages in
    cache exceed k bits,
    do the following twice for all j:
(7)     Evict(j, 2^{j+1}).
```

Figure 4: Randomized Online Algorithm for the Fault model.

The randomized algorithm for the fault model is shown in Figure 4. We will need the following lemma about the distribution of the number of bits RFMA has in the cache.

Lemma 9 *After each request is processed, for each l, the following three items depend only on the request sequence and are completely independent of the random choices made by the algorithm:*

1. *The set of marked pages.*
2. *The number of bits belonging to unmarked l-pages that RFMA has in its cache.*
3. *The beginning and ends of the l-phases.*

Proof. Let M_l denote the set of marked l-pages and let u_l denote the number of bits belonging to unmarked l-pages that RFMA has in its cache. We prove this lemma by induction on the number of requests. We will assume that the cache starts out empty, so the lemma is vacuously true after no pages are requested. Suppose that it is true for any sequence of t requests, and suppose we have a request to an l-page p at time $t+1$. p is brought into the cache and marked. M_l continues to be independent of any random choices made by the algorithm. By the inductive assumption, the event that p was requested in the previous l-phase is independent of the random choices made by the algorithm. If p was not requested in the previous phase, then RFMA did not have p in the cache and u_l remains unchanged. If p was requested in the previous phase and was unmarked, then at this point, the lemma may not be true. This is because $|p_{out}|$ bits may have been evicted and the number of bits from p that RFMA has in its cache does depend on the random choices made by the algorithm. Let u_l' denote the new value for u_l at line (4) of the algorithm after p is marked. $u_l' = u_l - |p|_{in}$. $|p|_{out} \leq u_l'$ is true if and only if $|p| \leq u_l$ which is independent of the random bits used by RFMA. If $u_l \leq |p|_{out}$, then u_l' goes to 0 after line (5). Otherwise, an addition $|p_{out}|$ bits will be evicted which means that u_l' will go down to $u_l - |p|$ which is again independent of the random choices made by the algorithm.

At step (6) of the algorithm, the number of bits the algorithm has in its cache is independent of the random bits used by the algorithm. If the size of the cache is exceeded, then we proceed to $\textsc{Evict}(j, 2^{j+1})$ for every j. By the inductive hypothesis, u_j is independent of the random bits used by the algorithm. If $u_j \geq 2^{j+1}$, the algorithm will evict 2^{j+1} bits from unmarked j-pages and u_j decreases by 2^{j+1}. If $u_j < 2^{j+1}$, the algorithm will evict u_j bits and a new phase begins. Denote the new number of bits belonging to unmarked pages by u'_j. The algorithm will then evict $\min\{u'_j, 2^{j+1} - u_j\}$ bits. Thus, the number of bits evicted and whether a new phase is started are still independent of the random choices made by the algorithm. Finally, if the number of unmarked bits only depends on the input sequence, the number of bits that are evicted in step (13) also only depends on the input sequence. □

For each class l, we normalize the size of the pages by dividing by 2^l. Thus, the normalized size of every page is less than 2 and at least 1. We will use the following three definitions in the proof:

- Let $d(l)$ denote the sum of the normalized sizes of the pages that were requested in the previous phase and are not currently marked (i.e. have not yet been requested in the current phase).
- Let $h(l)$ be the sum of the normalized sizes of the l-pages that the algorithm has evicted in the current phase that remain outside RFMA's cache. Note that since these pages were evicted in the current phase, they must have been requested in the previous phase or else they would not be in the cache in the first place. Since they are outside RFMA's cache, they are unmarked. Thus, $h(l) = d(l) - (u_l/2^l))$. (u_l was defined in the algorithm to be the number of bits belonging to unmarked l-pages that RFMA has in its cache).
- Let $m(l)$ be the sum of the normalized sizes of pages that were requested in the previous or current phase that are not in OPT's cache.

Note that if the algorithm has partially evicted a page p, p's contribution to $h(l)$ will be the fraction of p that has been evicted. Thus, if x bits of p have been evicted, then the contribution from p to $h(l)$ will be $x/2^l$. $d(l)$, $h(l)$ and $m(l)$ are all independent of the random choices made by the algorithm.

The following lemma will be key to the proof of competitiveness for RFMA:

Lemma 10 *Fix a page p that was requested in the previous phase that is not currently marked. The probability that any bits of p have been evicted is at most $2h(l)/d(l)$.*

Proof. At any given point, the probability that p is in the cache is the same as if the pages had been evicted according to the following process: pick a random permutation of the d pages that were requested in the previous phase and are currently unmarked. d is at least $d(l)/2$ since in the worst case, all the pages have size 2^{l+1}. Evict pages in this order until $h(l)$ bits have been evicted. What is the probability that a given page p will be evicted? The number of pages that will be evicted is at most $h(l)$. Thus, if p appears in the last $d - h(l)$ pages in the permutation, it will remain in the cache. The probability that p will remain in the cache is at least $(d - h(l))/d = 1 - h(l)/d \geq 1 - 2h(l)/d(l)$. This means that the probability that p is not in the cache is at most $2h(l)/d(l)$. □

We will use three potential functions per class:

- γ_l is the sum of the normalized sizes of marked l-pages that RFMA has in its cache and that OPT does not have in its cache.

- $$\phi_l = \begin{cases} 0 & \text{if all bits belonging to } l\text{-pages in RFMA's cache} \\ & \text{are marked or } m(l) - h(l) \leq 0 \\ \\ \max\{m(l) - h(l), 1/2\} & \text{otherwise} \end{cases}$$

- $\lambda_l = 4h(l)(H_{d(l)} - 1)$, where H_j is the j^{th} harmonic number.

Finally, we combine the potential functions for each class

$$\Phi = \sum_l \frac{1}{4}\phi_l \qquad \Gamma = \sum_l \frac{1}{4}\gamma_l \qquad \Lambda = \sum_l \lambda_l.$$

We prove the following lemma:

Lemma 11 *After each stage of the request:*

$$\textit{RFMA's cost } + \Delta\Lambda \leq ((64H_k + 16)\log k)[\textit{OPT's cost} - \Delta(\Phi + \Gamma)]. \tag{1}$$

Since Λ, Γ and Φ are initially 0 and are always non-negative, the lemma implies that RFMA is $((64H_k + 16)\log k)$-competitive. To see that λ_l is always non-negative, note that if $d(l)$ is ever 0, then $h(l)$ must also be 0 since $h(l) \leq d(l)$.

Proof. Consider a request to an l-page p at time t. We break the analysis into three cases:

Case 1: p is marked. RFMA already has the page in the cache. If OPT does not have p in the cache, it brings it into the cache. When OPT brings a page into the cache, $m(l)$ can only decrease which means that Φ and Γ can only decrease. If OPT then evicts a page from some class j, it incurs a cost of 1. $m(j)$ can increase by at most 2 which means that ϕ_j and γ_j can increase by at most two each. Thus, OPT's cost plus $\Delta(\Phi + \Gamma)$ is positive.

Case 2: p is not marked and was not requested in the previous phase. OPT brings p into the cache if it is not already there. $m(l)$ and hence Φ can only decrease. Then RFMA brings p into the cache and marks it. Since OPT already has p in the cache, neither γ_l, nor ϕ_l change. Using the same argument as the previous cases, OPT may evict a page in which case $\Phi + \Gamma$ does not decrease by more than OPT's cost.

If RFMA has enough room for page p, no evictions are performed and the next request is processed. If, however, there is not enough room for p, we will try and evict 2^{l+1} bits from class l for each l.

We first address what happens if in the process of evicting bits from class l, all the l-pages in the cache are marked and must be unmarked (i.e. a new l-phase begins). When all the l-pages become marked, ϕ_l goes to 0 which can only result in a decrease in ϕ_l. At this point, a new phase begins, so the set of pages currently in the cache become the set of pages requested in the previous phase. This also means that $h(l)$ becomes 0 since no bits have yet been evicted in the current phase for class l.

How does this change our potential functions? λ_l becomes 0 since $h(l)$ is now 0. This means that λ_l does not increase. The set of marked l-pages at the end of the old l-phase are exactly those l-pages that RFMA has in its cache. This means that the sum of the normalized sizes of pages that RFMA has in the cache and OPT does not have in the cache is exactly γ_l. When all the pages become unmarked, γ_l goes to 0. The old value of ϕ_l is 0. The new value of ϕ_l is $m(l)$ which is the sum of the normalized sizes of pages that RFMA has in the cache that OPT does not have in the cache. Note that this new value is either 0 or at least 1, so the 'max' in the definition of ϕ_l has no effect. Thus, γ_l decreases by the amount by which ϕ_l increases and $\Delta(\Phi + \Gamma) = 0$, and all the changes to the potential functions brought about by a new phase obey Inequality 1.

Now RFMA attempts to evict 2^{l+1} bits from each class. Since we are charging RFMA as soon as it evicts any bits from a given pages, this will result in at most a cost of two per class. If any bits are evicted in step (13) of $\textsc{Evict}(l, C)$, this will not cost the algorithm anything because this has to be a partially evicted page. For each class l, $h(l)$ increases by at most two. Thus, each λ_l increases by at most $8(H_{d(l)} - 1)$, and the total amortized cost to RFMA is

$$2 \log k + 8 \sum_{l=1}^{\log k} (H_{d(l)} - 1) \leq (8H_k + 2) \log k.$$

We now prove that Φ decreases by at least $1/8$.

Since RFMA has exceeded the capacity of its cache, it must be the case that for some class l, RFMA has more bits belonging to l-pages in the cache than does OPT. Pick one such l. Note that RFMA must have at least 2^l bits belonging to unmarked l-pages in the cache or else they would have been evicted in step (13) of $\textsc{Evict}(l, C)$.

Let a be the sum of the normalized sizes of the l-pages that were requested in the current or previous phase. The number of bits belonging to l-pages that RFMA has in its cache is $a - h(l)$. Let b be the sums of the normalized sizes of the l-pages that OPT has in its cache. We know that $a - h(l) > b$. Thus, $m(l)$, which is at least $a - b$, is greater than $h(l)$. This means that $m(l) - h(l) > 0$.

When the l-pages are evicted, $h(l)$ increases by at least one and thus ϕ_l decreases by a non-negative amount. If the value of $m(l) - h(l)$ afterwards is negative, then ϕ_l decreases by at least $1/2$ (since it was at least $1/2$ before and is 0 now). If $m(l) - h(l)$ is positive afterwards, then it was at least 1 before the eviction. In this case, ϕ_l also decreases by at least $1/2$. Since all the other evictions can only serve to decrease Φ, we know that Φ decreases by at least $1/8$.

Case 3: p is unmarked and was requested in the previous phase. OPT brings p into the cache if it is not already there. $m(l)$ and hence Φ can only decrease. Then p is marked and $\min\{u_l, |p_{out}|\}$ bits are evicted. Since OPT has p in its cache, γ_l does not change. Either $|p|_{out}$ bits belonging to l-pages are evicted or all the remaining bits belonging to unmarked l-pages are evicted. In the former case, $h(l)$ does not change and ϕ_l is unchanged. In the latter case, ϕ_l becomes 0 which means that it does not increase. As in case 1, if OPT evicts a page, the cost that OPT incurs is at least as large as the amount by which Φ decreases.

Finally, we examine the amortized cost of the algorithm. A page is evicted only if p was not already in the cache. By Lemma 10, this happens with probability at most $2d(l)/h(l)$. The cost in evicting $|p|_{out}$ bits is at most 2. Thus, the expected cost is at most $4h(l)/d(l)$. $h(l)$ does not increase and $d(l)$ decreases by one. Thus, the change to Λ is at most

$$4h(l)[(H_{d(l)-1} - 1) - (H_{d(l)} - 1)] = -\frac{4h(l)}{d(l)}.$$

If at this point, all the l-pages become marked, then ϕ_l goes to zero. This can only amount to a decrease in ϕ_l since it is always non-negative. If the size of memory, is still exceeded, RFMA will go to step (6) of the algorithm. The analysis is the same as in Case 2.

□

5.2 The BitModel.

The randomized algorithm for the bit model, which we call the *Randomized Counter Algorithm* (RCA), is given in Figure 5.2. A counter is maintained for each class. $c(l)$ is the current value of the counter for class l and is initialized to 0 at the beginning of the algorithm. The structure of the proof for the randomized algorithm in the bit model is much the same as the structure of the proof for the fault model, except for the portions that use the counter which are similar to the proof for the offline bit model algorithm.

```
Algorithm RandomizedCounterAlgorithm:
Consider a request to an $l$-page $p$ of size $x$:
(1) Bring $p$ into the cache if not already there.
(2) Mark page $p$.
(3) if $p$ was requested in the previous phase,
(5)     Let $u_l$ be the number of bits belonging to
        unmarked $l$-pages in the cache.
(6)     Evict$(l, \min\{u_l, |p|_{out}\})$.
(7) if the number of bits that RCA has in the cache
    exceeds $k$ by $e > 0$:
        do the following:
(8)     if the total sizes of pages that RCA has in the cache
        belonging to classes $cs(e)$ or below is less than $e$
            Let $r$ be the lowest class greater than $cs(e)$ such that
            RCA has at least a portion of an $r$-page in its cache.
(9)         $m \leftarrow 2^r$.
(10)    else $m \leftarrow e$
(11)    For all $j \leq cs(e)$
(12)            Evict$(j, m)$
(13)    For all $j > cs(e)$
(14)        $c(j) \leftarrow c(j) + m$;
(15)        while $c(j) \geq 2^j$ and there are $j$-pages in the cache,
(16)            Evict$(j, 2^j)$
(17)            $c(j) \leftarrow c(j) - 2^j$.
```

Figure 5: Randomized Online Algorithm for the Bit Model.

We will need the following lemma about the distribution of the number of bits RFMA has in the cache.

Lemma 12 *After each request is processed, for each l, the following four items depend only on the request sequence and are completely independent of the random choices made by the algorithm:*

1. *The set of marked pages.*
2. *The number of bits belonging to unmarked l-pages that RFMA has in its cache.*
3. *The beginning and ends of the l-phases.*
4. *$c(l)$.*

Proof. The proof is almost the same as the proof for Lemma 9 with the added observation that since m depends only on values that are independent of the random choices made by the algorithm it is also independent of the random choices made by the algorithm. Thus, any changes to $c(l)$ are also independent of the random choices made by the algorithm. □

We will use the following three definitions in the proof:

- Let $d(l)$ denote the sum of the sizes of the l-pages that were requested in the previous phase and are not currently marked (i.e. have not yet been requested in the current phase).
- Let $h(l)$ be the sum of the sizes of l-pages that the algorithm has evicted in the current phase that remain outside RCA's cache. Note that since these pages were evicted in the current phase, they must have been requested in the previous phase or else they would not be in the cache in the first place. Since they are outside RFMA's cache, they are unmarked. Thus, $h(l) = d(l) - u_l$. (u_l is defined in the algorithm to be the number of bits belonging to unmarked l-pages that RCA has in the cache).
- Let $m(l)$ be the sum of the sizes of l-pages that were requested in the previous or current phase that are not in OPT's cache.

Note that if the algorithm has partially evicted a page p, p's contribution to $h(l)$ will be the fraction of p that has been evicted. Thus, if x bits of p have been evicted, then the contribution from p to $h(l)$ will be x. $d(l)$, $h(l)$ and $m(l)$ are all independent of the random choices made by the algorithm.

The following lemma will be key to the proof of competitiveness for RCA:

Lemma 13 *Fix a page p that was requested in the previous phase that is not currently marked. The probability that any bits of p have been evicted is at most $2h(l)/d(l)$.*

Proof. The proof is almost the same as the proof for Lemma 10, except that since the definitions are in terms of the actual sizes of the pages instead of the normalized sizes of the pages, d is at least $d(l)/2^{l+1}$ and the number of pages that have been partially evicted is at most $h(l)/2^l$. □

We will use three potential functions per class:

- γ_l is the sum of the normalized sizes of marked l-pages that OPT does not have in its cache.

- $$\phi_l = \begin{cases} 0 & \text{if all bits belonging to } l\text{-pages in cache are marked or } m(l) - h(l) \leq 0 \\ \max\{m(l) - h(l), 2^{l+2}/5\} - c(l)/5 & \text{otherwise} \end{cases}$$

- $\lambda_l = 4h(l)(H_{\lceil d(l)/2^l \rceil} - 1) + 2c(l)(H_k - 1)$, where H_j is the j^{th} harmonic number.

Note that when m is added to $c(l)$ in step (13), it is always the case that $m \leq 2^l$. This means that $c(l) \leq 2^l$ after every request is processed and ϕ_l is always non-negative.

Finally, we combine the potential functions for each class

$$\Phi = \sum_l \frac{1}{2}\phi_l \qquad \Gamma = \sum_l \frac{1}{2}\gamma_l \qquad \Lambda = \sum_l \lambda_l.$$

We prove the following lemma:

Lemma 14 *After each stage of the request:*

$$\textit{RCA's cost } + \Delta\Lambda \leq ((25H_k + 20)\log k)[\textit{OPT's cost} - \Delta(\Phi + \Gamma)]. \tag{2}$$

Since Λ, Γ and Φ are initially 0 and are always non-negative, the lemma implies that RRFMA is $((25H_k + 20)\log k)$-competitive. To see that λ_l is always non-negative, not that if $d(l)$ is ever 0, then $h(l)$ must also be 0 since $h(l) \leq d(l)$.

Proof. Now consider a request to an l-page p at time t. We break the analysis into three cases:

Case 1: p is marked. RFMA already has the page in the cache. If OPT does not have p in the cache, it brings it into the cache. When OPT brings a page into the cache, $m(l)$ can only decrease which means that Φ and Γ can only decrease. If OPT evicts an l-page p of size $|p|$, it incurs a cost of $|p|$. $m(l)$ and hence ϕ_l and γ_l can increase by at most $|p|$ each. Thus, OPT's cost plus $\Delta(\Phi + \Gamma)$ is positive.

Case 2: p is not marked and was not requested in the previous phase. OPT brings p into the cache if it is not already there. $m(l)$ and hence Φ can only decrease. Then RCA brings p into the cache and marks it. Since OPT already has p in the cache, neither γ_l, nor ϕ_l change. Using the same argument as the previous cases, OPT may evict a page in which case $\Phi + \Gamma$ does not decrease by more than OPT's cost.

If RCA has enough room for page p, no evictions are performed and the next request is processed. If, however, there is not enough room for p, the algorithm proceeds to step (8).

We first address what happens if in the process of evicting bits from class l, all the l-pages in the cache are marked and must be unmarked (i.e. a new l-phase begins). When all the l-pages become marked, ϕ_l goes to 0 which can only result in a decrease in ϕ_l. At this point, a new phase begins, so the set of pages currently in the cache become the set of pages requested in the previous phase. This also means that $h(l)$ becomes 0 since no bits have yet been evicted in the current phase for class l.

How does this change our potential functions? Since $h(l)$ goes to 0 and was non-negative before, λ_l does not increase. The set of marked l-pages at the end of the old l-phase are exactly those

l-pages that RFMA has in its cache. This means that the sum of the sizes of pages that RFMA has in the cache and OPT does not have in the cache is exactly γ_l. When all the pages become unmarked, γ_l goes to 0. The old value of ϕ_l is 0. The new value of ϕ_l is $m(l)$ which is the sum of the sizes of pages that RFMA has in the cache and that OPT does not have in the cache. Note that this new value is either 0 or at least 2^l, so the 'max' in the definition of ϕ_l has no effect. Thus, γ_l decreases by the amount by which ϕ_l increases and $\Delta(\Phi + \Gamma) = 0$, and all the changes to the potential functions brought about by a new phase obey Inequality 2.

Now, the algorithm attempts to evict m bits from each l-class.

For $j \leq cs(e)$, the algorithm will evict at most m bits from class j. Since we charge the algorithm for evicting an entire page whenever it evicts the first bit for a page, the algorithm will incur a cost of at most $m + 2^{j+1}$. $h(j)$ increases by at most m. Thus, the total amortized cost is at most $(5m + 2^{j+1})H_{d(l)}$. For $j > cs(e)$, we add m to $c(j)$. Thus, the total amortized cost is at most $2mH_k$. The total amortized cost for all classes is

$$\begin{aligned} & \sum_{j \leq cs(e)} (5m + 2^{j+1}) H_{\lceil d(j)/2^j \rceil} + \sum_{j > cs(e)} 2mH_k \\ \leq\ & 5m \log k H_k + \sum_{j \leq cs(m)} 2^{j+1} H_k \\ \leq\ & 5m \log k H_k + 4mH_k = (5 \log k + 4) H_k m \end{aligned}$$

If any additional bits are evicted in step (13) of $\textsc{Evict}(l, C)$, they do not cost the algorithm since those bits must belong to a partially evicted page.

We must now prove that Φ decreases by at least $m/5$.

Denote by $b_{OPT}(j)$ the sum of the sizes of the j-pages that OPT has in its cache. Similarly, $b_{RCA}(j)$ is the sum of the sizes of the j-pages that RCA has in its cache. Let $e(j) = b_{RCA}(j) - b_{OPT}(j)$. Let S be the sum of the sizes of the pages that have been requested in the previous or the current phase. We have that $m(j) \geq S - b_{OPT}(j)$ and $S - h(l) = b_{RCA}(j)$. Putting these together, we get that $m(j) - h(j) \geq e(j)$. We make the following claim:

Claim 15 *For $j \leq cs(e)$, if $e(j) > 0$, then ϕ_j decreases by at least $e(j)/5$. For $j > cs(e)$, if $e(j) > 0$, then ϕ_j decreases by at least $m/5$.*

In either case, as long as $e(j) > 0$, ϕ_j decreases by at least $e(j)/5$, since $m \geq e \geq e(j)$. The claim will be sufficient to establish that Φ decreases by at least $m/5$ because if m was chosen to be e, then we have that

$$\sum_{e(j)>0} e(j) \geq \sum_j e(j) = e = m,$$

and as long as ϕ_j decreases by $e(j)/5$ for each class with $e(j) > 0$, the total decrease to Φ is at least $m/5$. Note that ϕ_j does not increase for any j when RCA, so we need not be concerned with those classes j for which $e(j) \leq 0$.

Alternatively, suppose m was chosen in step (9). Let r be the lowest class greater than $cs(e)$ such that RCA has at least a portion of an r-page in its cache. We know that the sum of the sizes of the pages in classes of size $cs(e)$ or less are not enough to compensate for the excess that RCA has in the cache. Thus, for some $j > cs(e)$, the sum of the sizes of the j-pages that RCA has in cache is more than OPT. Thus, we are guaranteed that $e(j) > 0$ for some $j \geq cs(m)$. Then, by the claim, ϕ_j decreases by at least $m/5$.

Now we must prove the claim. We break proof of the claim into two cases:

$j > cs(e)$: First $c(j)$ is incremented by m. Note that since the algorithm has at least some portion of a page from class j in the cache, $j \geq cs(m)$. This implies that $m < 2^{j+1}$. Since $e(j) > 0$, we know that $m(j) - h(j) > 0$. Since $c(j) < 2^{j+1}$ before $c(j)$ is incremented, $\phi_j > 2^{j+1}/5$ before $c(j)$ is incremented. This means that when m is added to $c(j)$, ϕ_j does in fact decrease by $m/5$.

Now suppose that the condition in step (15) is true. $c(j)$ is decremented by 2^j. The number of unmarked bits belonging to j-pages in the cache is either 0 or at least 2^j because otherwise, they would be evicted in step (13) of Evict. This means that 2^j are in fact evicted and $h(j)$ increases by 2^j. Since $h(j)$ increases by the amount that $c(l)$ decreases, the change to λ_l is at most 0. Now we determine the change to ϕ_l. If after the eviction, $m(j) - h(j)$ becomes 0, then ϕ_j also becomes 0. Since ϕ_j is always non-negative, it certainly has not increased. Now suppose that $m(j) - h(j) > 0$ after the eviction. This means that $\max\{m(j) - h(j), 2^{j+2}/5\}$ decreases by at least $2^j - 2^{j+2}/5$ and ϕ_j decreases by at least $(2^j - 2^{j+2}/5) - 2^j/5$. Note that this value is non-negative.

$j \leq cs(e)$: Suppose that $m(j) - h(j) = 0$ after the evictions. ϕ_j decreases by at least

$$\max\left\{m(j) - h(j), \frac{2^{j+2}}{5}\right\} - \frac{2^{j+1}}{5} \geq \frac{m(j) - h(j)}{5} \geq \frac{e(j)}{5}.$$

Now suppose that $m(j) - h(j) > 0$ after the evictions. This means that the number of bits belonging to j-pages that RCA has in the cache after the evictions is non-zero and the algorithm successfully evicted m bits. We also know that $m \geq 2^j$ since $cs(m) \geq cs(e) \geq j$. Thus ϕ_j decreases by at least

$$m - \frac{2^{j+2}}{5} > \frac{m}{5} \geq \frac{e(j)}{5}.$$

Case 3: p is unmarked and was requested in the previous phase. OPT brings p into the cache if it is not already there. $m(l)$ and hence Φ can only decrease. Then p is marked and $\min\{u_l, |p|_{out}\}$ bits are evicted. Since OPT has p in the cache, γ_l does not change. If p was complete in the cache, then $h(l)$ does not change. If $|p|_{out}$ bits of p had been evicted, then after p is marked, $h(l)$ decreases by $|p|_{out}$. In this case, either $|p|_{out}$ bits belonging to l-pages are evicted or all the remaining bits belonging to unmarked l-pages are evicted. In the former case, $h(l)$ does not change and ϕ_l in unchanged. In the latter case ϕ_l becomes 0 which means that it does not increase. As in case 1, if OPT evicts a page, the cost that OPT incurs is at least as large as the amount by which Φ decreases.

Finally, we examine the left hand side of Inequality 2. A page is evicted only if p was not already in the cache. By Lemma 10, this happens with probability at most $2d(l)/h(l)$. The cost in evicting $|p|_{out}$ bits is at most 2^{l+1}. Thus, the expected cost is at most $2^{l+2}h(l)/d(l)$. $h(l)$ does not increase and $d(l)$ decreases by at least 2^l. Thus, the change to Λ is at most

$$4h(l)[(H_{\lceil (d(l)-2^l)/2^l \rceil} - 1) - (H_{\lceil d(l)/2^l \rceil} - 1)] = -\frac{2^{l+2}h(l)}{d(l)}.$$

If at this point, all the l-pages becomed marked, then ϕ_l goes to zero. This can only amount to a decrease in ϕ_l since it is always non-negative. If the size of memory, is still exceeded, RFMA will go to step (6) of the algorithm. The analysis is the same as in Case 2.

```
Algorithm FaultModelDominatingDistributionAlgorithm:
Consider a request to l-page p:
(1) Bring p into the cache if not already there.
(2) if there are more than k bits in the cache,
(3)     For all j:
(4)         if there are any j-pages in the cache
(5)             if there is only one j-page, evict it.
(6)             else evict a pair of pages chosen according to the
                pairs dominating distribution.
```

Figure 6: Online algorithm under the fault model when the sequence is generated according to a known distribution.

6 Probabilistic Analysis

Now we turn to the scenario where there is a distribution over the sequence known to the algorithm in advance. Although from an information-theoretic point of view, it is possible to find the algorithm that minimizes the expected cost, it is not feasible to find this optimal algorithm. It is shown by Karlin, Phillips and Raghavan that even in the case of paging with uniform size pages, the problem of computing the optimal online strategy when the sequence is generated by a markov chain, is a linear program in $n\binom{n}{k}$ variables, where n is the total number of pages [KPR92]. Thus, we seek an approximation algorithm that will come within some factor of the best online algorithm. The algorithm that we present here is for the Fault model and is combinations of the approximations for the offline case and an algorithm due to Lund, Phillips and Reingold for paging with uniform pages under a known distribution [LPR94].

At any moment in time, given any two pages p and q, one can determine from the distribution over the remainder of the sequence the probability that p is requested before q. Lund *et al.* prove that for any set S of pages, there is a distribution over S (called the *dominating distribution*) such that if p is chosen according to the dominating distribution, for any $q \in S$, the probability that q appears at least as soon as p is at least $1/2$. The dominating distribution, of course, depends heavily on the pairwise probabilities that a given page p_1 is requested before another page p_2. Since these pairwise probabilities are available to the algorithm, the dominating distribution can be computed.

We use a variant of the dominating distribution in which two pages are chosen instead of one. We call this distribution the *pairs dominating distribution.* Let V be the set of pages. Let V_2 be the set of all sets of two pages. We define a function $w : V \times V_2 \to [0,1]$ with the property that if $a \in \{b,c\}$, then $w(a,\{b,c\}) = 0$. Furthermore, if a, b and c are all distinct, then $w(a,\{b,c\}) + w(b,\{a,c\}) + w(c,\{a,b\}) = 2$. A pairs dominating distribution for V and w is a probability distribution p over V_2 such that if $\{a,b\}$ are chosen according to p, then for each $c \in V$, $E[w(c,\{a,b\})] \leq 1/2$. In our case, $w(c,\{a,b\})$ is the probability that $c \notin \{a,b\}$ and c is not requested before both a and b. The proof that a pairs dominating distribution always exists appears in the appendix. If the algorithm is asked to evict two pages from class j, it will always choose according to the pairs dominating distribution over the j-pages that it has in its cache.

Figure 6 gives a randomized algorithm for the fault model called the Fault Model Dominating Distribution Algorithm that works when the input is generated according to an arbitrary distribution. We will prove the following theorem:

Theorem 16 *Let $\mathcal{D}$ be a distribution over request sequences. Let $\texttt{cost}_A(\mathcal{D})$ be the cost under the fault model of an online algorithm A when the input sequence is generated according to $\mathcal{D}$. We prove that for any online algorithm A,*

$$\texttt{cost}_{FMDD}(\mathcal{D}) \leq (8 \log k)\texttt{cost}_A(\mathcal{D}).$$

Proof. If step 2 is entered, then the algorithm performs at most $2 \log k$ evictions. We will prove that the expected number of times that the algorithm enters step (2) is at most 8 times the expected number of faults of any online algorithm that knows the distribution over the request sequence. In particular, we will compare $FMDD$ to an arbitrary online algorithm ON. Let $S_{FMDD}(l)$ denote the set of l-pages that FMDD has in the cache. Similarly for $S_{ON}(l)$.

We will use a variation of the accounting scheme developed in [LPR94] for the proof of the dominating distribution algorithm for uniform size pages. We will maintain a mapping c. Let P be the set of all pages. Let P_2 be the set of all sets of two pages. c is defined over a subset of $P \cup P_2$. Here's how we maintain the mapping c:

Consider a request to page s. Both ON and FMDD bring s into the cache. ON may evict any number of pages (including s). Now if FMDD has more than k bits in the cache, it will evict two pages from every class. We know that before it performs these evictions, there is some class j for which the sum of the sizes of the j-pages that FMDD has in the cache is more than the sum of the size of the j-pages that ON has in its cache. Let q and q' be the j-pages that FMDD evicts. If $q \notin S_{ON}(j)$, then $c(q) = q$. Else if $q' \notin S_{ON}(j)$, then $c(q') = q'$. If both q and q' are in $S_{ON}(j)$, then we will prove that there is some page $p \in S_{FMDD}(j) - S_{ON}(j)$ to which nothing is mapped. We then set $c(\{q, q'\}) = p$. If at any point ON evicts q, then $c(\{q, q'\})$ becomes undefined and $c(q) = q$. (Similarly for q'). If $c(\{a, b\})$ is defined and either a or b is requested, then $c(\{a, b\})$ becomes undefined. Similarly, if $c(a)$ is defined, and a is requested then $c(a)$ becomes undefined.

The following facts are easy to verify by induction.

1. If $c(p)$ is defined, then $c(p) = p$.

2. If $c(p) = p$, then $p \notin S_{FMDD} \cup S_{ON}$.

3. If $c(\{a, b\})$ and $c(\{x, y\})$ are defined, then $\{a, b\} \cap \{x, y\} = \emptyset$.

4. If $c(\{a, b\})$ is defined, then $c(a)$ and $c(b)$ are not defined.

5. If $c(\{a, b\})$ is defined, then $c(\{a, b\}) \in S_{FMDD}$ and $\{a, b\} \subseteq S_{ON} - S_{FMDD}$.

Now examine the moment just before q and q' are evicted. At this moment, the number of bits in $S_{FMDD}(l)$ is strictly larger than the number of bits in $S_{ON}(l)$. By Facts (1) and(2) above, if $c(X) = s$ for some $X \in P \cup P_2$ and $s \in S_{FMDD}(l)$, then X is a pair $\{a, b\}$. Furthermore by Fact (5), $\{a, b, \} \subseteq S_{ON}(l) - S_{FMDD}(l)$. This means that for every page s in $S_{FMDD}(l)$ to which something is mapped, there is a pair of pages in $S_{ON}(l)$ that maps to s. Furthermore, by Fact (3), all these pairs are disjoint. Since the sum of the sizes of a and b is at least the size of $c(\{a, b\})$ and $S_{FMDD}(l) \geq S_{ON}(l)$, it must be the case that there is an $s \in S_{FMDD}(l) - S_{ON}(l)$ to which nothing is mapped.

Let $\chi(t) = 1$ if and only if FMDD makes an assignment at time t. FMDD incurs a cost of at most $2 \log k$ per request. Furthermore, for every request on which is incurs a cost, it will make an assignment. This means that The cost to FMDD is:

$$C_{FMDD} \leq 2 \log k \sum_t \chi(t).$$

Suppose that FMDD makes an assignment to $c(X) = s$ at time t. We will say that the assignment is *good* if $c(X)$ is requested on or before the first time any page in X is requested. There are two possibilities in this case. The first is that $c(X)$ remains equal to s at the time it is requested. In this case, the request to s causes the assignment to disappear. The second is that ON evicts some page p in X before s is requested in which case the assignment is moved to $c(p) = p$. In this case, the next request to p makes the assignment disappear. In either case, ON will incur a cost on the request that causes the assignment to disappear since the requested page is not in ON's cache. Let $\gamma(t) = 1$ iff FMDD makes a good assignment to $c(X)$ at time t. Since only two elements in $P \cup P_2$ can be assigned to the same $s \in P$, each request in which ON incurs a cost can make at most two assignments disappear and we have that

$$C_{ON} \geq \frac{1}{2} \sum_t \gamma(t).$$

We can think of assigning $c(\cdot)$ as follows. Pick an $s \in S_{ON} - S_{FMDD}$ to which nothing is mapped. Then the pair $\{q, q'\}$ is chosen according to the dominating distribution. If $s \in \{q, q'\}$, then $c(s) = s$ and it is definitely the case that $\gamma(t) = 1$. If $s \notin \{q, q'\}$, then the probability that s is requested after the first request to q or q' is at least $1/2$ by the fact that $\{q, q'\}$ was chosen according to a dominating distribution. Thus, we have that that $E[2\gamma(t)] \geq E[\chi(t)]$. Putting the inequalities, we get that

$$8 \log k E[C_{ON}] \geq E[C_{FMDD}].$$

□

Acknowledgements: The author would like to thank Anja Feldman, Steven Phillips, Anna Karlin and David Karger for many useful discussions and for bringing this problem to her attention.

References

[AAK99] S. Albers, S. Arora and S. Khanna. Page Replacement for General Caching Problems. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 31–40, 1999.

[AW96] M.F. Arlitt and C.L. Williamson. Web server workload characterization: The search for invariants. *Proceedings of SIGMETRICS*, Philadelphia, PA, April, 1996.

[Bel66] L.A. Belady. A study of replacement algorithms for virtual storage computers. *IBM Systems Journal*, 5:78–101, 1966.

[BCC+95] A. Betsavros, R. Carter, M. Crovella, C. Cunha, A. Heddaya, S. Mirdad. Application-level document caching in the internet. *Proceedings of the Second International Workshop on Services in Distributed and Networked Environments*, pp. 166-173, June 1995.

[CI97] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pp. 193–206, 1997.

[CK99] E. Cohen and H. Kaplan. LP-based analysis of greedy-dual size. *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 879–880, 1999.

[DHS93] P.B. Danzig, R.S. Hall and M.F. Schwartz. A case for caching file objects inside internetworks. *Proceedings of ACM Sigcomm*, pp. 239 - 248, September 1993.

[Fiat97] A. Fiat. Private Communication.

[FKIP96] A. Feldman, A. Karlin, S. Irani, S. Phillips. Private Communication.

[FKL+91] A. Fiat, R. Karp, M. Luby, L. A. McGeoch, D. Sleator, and N.E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[KPR92] Karlin, A.R., Phillips, S.J, Raghavan, P. Markov Paging In *33rd IEEE Symposium on Foundations of Computer Science*, pages 208–217, 1992.

[KMRS88] Karlin, A.R., Manasse, M.S., Rudolph, L., Sleator, D.D. Competitive snoopy caching. *Algorithmica*, 1988, vol.3, (no.1):79-119.

[KMR95] T.T. Kwan, R.E. McGrath, and D.A. Reed. NCSA's World Wide Web server: design and performance. *IEEE Computer*, pp 68 - 74, November, 1995.

[LPR94] C. Lund, S. Phillips, and N. Reingold. Ip over connection-oriented networks and distributional paging. In *35th IEEE Symposium on Foundations of Computer Science*, pages 424–435, 1994.

[M96] E.P. Markatos. Main memory caching of web documents. *World Wide Web Conference*, Paris, France, May, 1996.

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

[T85] R. E. Tarjan. Amortized Computational Complexity. *SIAM Journal on Discrete Mathematics*, vol. 6, (no. 2), 1985.

[WAS+96] A. Williams, M. Abrams, C.R. Stanbridge, G. Abdulla, and E.F. Fox. Removal policies in network caches for world-wide web documents. *ACM: Sigcomm96*, August, 1996.

[Young98] N. Young. Online file caching. *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 82–86, 1998.

[Young94] N. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11:525-541, 1994.

Appendix

In this section, we prove that a pairs dominating distribution exists for any set of pages. Let V be any finite set and V_2 be the set of all sets of size two subsets of V. We define a function $w : V \times V_2 \to [0,1]$ with the property that if $a \in \{b,c\}$, then $w(a,\{b,c\}) = 0$. Furthermore, if a, b and c are all distinct, then $w(a,\{b,c\}) + w(b,\{a,c\}) + w(c,\{a,b\}) = 2$. A pairs dominating distribution for V and w is a probability distribution p over V_2 such that if $\{a,b\}$ are chosen according to p, then for each $c \in V$, $E[w(c,\{a,b\})] \leq 1/2$.

Lemma 17 *Given any set V and function w with the properties described above, there is a pairs dominating distribution for (V,w).*

The following technical lemma will be useful:

Lemma 18 *Let V be any finite set and let p be a distribution over V. Then*

$$\frac{1-\sum_{v\in V}(p(v))^3}{1-\sum_{v\in V}(p(v))^2} \leq \frac{3}{2}.$$

Proof. We start with the case that there are only two elements v_1 and v_2 in V with non-zero probability. Let $p(v_1) = x$. Then $p(v_2) = 1-x$ and $p(v) = 0$ for any $v \notin \{v_1, v_2\}$. In this case:

$$\frac{1-\sum_{v\in V}(p(v))^3}{1-\sum_{v\in V}(p(v))^2} = \frac{1-x^2-(1-x)^2}{1-x^3-(1-x)^3} = \frac{3}{2}.$$

Now suppose that we start with some distribution p and change is by subtracting some weight from an element v_i with non-zero weight and add it to an element v_j with 0 weight. That is, $p'(v) = p(v)$ for $v \notin \{v_i, v_j\}$, $p'(v_i) = p(v_i) - \delta$ and $p'(v_j) = p(v_j) + \delta = \delta$. How does this change the quanity in question?

The numerator increases by $3p(v_i)^2\delta - 3p(v_i)\delta^2$. The denominator increases by $2\delta p(v_i)$. Note that the ratio of the changes is at most $3/2$. Thus, if the current quantity in question is at most $3/2$ and we make a change such that the ratio of the change to the numerator and the change to the denominator is bounded by $3/2$, then the resulting quantity is also bounded by $3/2$.

Finally, we can achieve any distribution q, by starting with $p(v_1) = q(v_1)$ and $p(v_2) = \sum_{i=2}^{n} q(v_i)$ and for $i = 3, \ldots, n$, moving $q(v_i)$ weight from $p(v_2)$ to $p(v_i)$. $\square$

Proof of Lemma17. Consider the following linear program: minimize x subject to

$$\sum_{\{a,b\}\in V_2} w(c,\{a,b\})p(\{a,b\}) \leq x \quad \text{for all } c \in V$$

$$\sum_{\{a,b\}\in V_2} p(\{a,b\}) = 1$$

$$p(\{a,b\}) \geq 0 \quad \text{for all } \{a,b\} \in V_2$$

The claim is that the solution to the linear program is at most $1/2$. The dual linear program is to maximize y subject to:

$$\sum_{c\in V} w(c,\{a,b\})q(c) \geq y \quad \text{for all } \{a,b\} \in V_2$$

$$\sum_{c \in V} q(c) = 1$$

$$q(c) \geq 0 \quad \text{for all } c \in V$$

It suffices to show that for any distribution q, there is a $\{a,b\} \in V_2$ such that

$$\sum_{c \in V} w(c, \{a,b\}) q(c) \leq \frac{1}{2}.$$

To this end, consider:

$$\begin{aligned}
& \sum_{\{a,b\} \in V_2} q(a)q(b) \sum_{c \in V} w(c, \{a,b\}) q(c) \\
= & \sum_{\{a,b\} \in V_2} \sum_{c \in V} q(a)q(b)q(c) w(c, \{a,b\}) \\
= & \sum_{a,b,c \in V | a<b<c} q(a)q(b)q(c) [w(c, \{a,b\}) + w(b, \{a,c\}) + w(a, \{c,b\})] \\
= & \sum_{a,b,c \in V | a<b<c} 2q(a)q(b)q(c)
\end{aligned}$$

We will also use the following inequality:

$$1 = \left(\sum_{c \in V} q(c) \right)^3 \geq \sum_{c \in V} q(c)^3 + 6 \sum_{a,b,c \in V | a<b<c} q(a)q(b)q(c).$$

Putting these together, we get that

$$\begin{aligned}
& \frac{\sum_{\{a,b\} \in V_2} q(a)q(b) \sum_{c \in V} w(c, \{a,b\}) q(c)}{\sum_{\{a,b\} \in V_2} q(a)q(b)} \\
= & \frac{\sum_{\{a,b\} \in V_2} q(a)q(b) \sum_{c \in V} w(c, \{a,b\}) q(c)}{1 - \sum_{a \in V} q(a)^2} \\
= & \frac{\sum_{a,b,c \in V | a<b<c} 2q(a)q(b)q(c)}{1 - \sum_{a \in V} q(a)^2} \\
\leq & \left(\frac{1}{3} \right) \frac{1 - \sum_{c \in V} q(c)^3}{1 - \sum_{a \in V} q(a)^2} \\
\leq & \frac{1}{2}
\end{aligned}$$

The last inequality comes from the technical lemma above. Thus, we know that for a weighted average over $\{a,b\}$, $\sum_{c \in V} w(c, \{a,b\}) q(c)$ is bounded by $1/2$. This means that there must be some $\{a,b\}$ for which $\sum_{c \in V} w(c, \{a,b\}) q(c) \leq 1/2$. $\square$