Caching with Expiration Times

Parikshit Gopalan* Howard Karloff† Aranyak Mehta‡ Milena Mihail§
Nisheeth Vishnoi¶

Abstract

Caching data together with expiration times beyond which the data is no longer valid is a standard method for promoting information consistency in distributed systems, including the Internet and WWW, large databases, and mobile telecommunications. We use the framework of competitive analysis of online algorithms and study page eviction strategies in the case where data has expiration times. We show that suitable adaptations of LRU and its generalizations, namely marking algorithms, are asymptotically optimal and, in the worst case, within a multiplicative factor 2 of the lower bounds. A key technical ingredient of our analysis is a covering invariant that captures some of the subtleties introduced by expiration times. The additional difficulty of dealing with expiration times is also reflected in our analysis of the randomized online marking algorithm, as well as the offline version of the problem for which we obtain a factor 3 approximation. We complement our theoretical findings with experiments on real and synthetic data.

1 Introduction

Caching data together with expiration times beyond which the data is no longer valid is a standard mechanism for promoting information consistency in distributed systems. It has been considered in the context of distributed memory in computer architecture [19], in distributed databases and datawarehouses [1, 10, 17, 23], in a wide range of Internet and WWW applications [3, 4, 7, 13, 14, 18, 27, 28, 29, 30] and in mobile and personal communications services [8, 9, 10]. For example, a WWW server may cache pages together with an estimate of the time before the information in the page becomes stale (also known as TTL "time-to-live"), expiration times are used to promote consistency in DNS (Domain Name Server) resolution, while in advanced telephony expiration times are used, among others, to maintain correct routing for mobile services and valid translations of 800-numbers.

So far, the typical question asked in the distributed information applications mentioned above has been "what to cache" rather than "how to cache". "What to cache" corresponds to prefetching policies, while "how to cache" raises eviction policies. For example, in the WWW context, the most important issue has been "among the whole Web, what is useful to prefetch, and how can expiration times be assigned so that the information does not become stale"? Addressing this question, it is by now well accepted that there are several forms of prefetching which improve performance [11, 12, 18, 28, 29, 30]. Moving one step ahead, once intelligent cache population mechanisms have been established, the next question becomes "how shall we cache"? In particular, when eviction becomes necessary, the question becomes "what shall we evict"?

In addition, the problem of eviction when pages have expiration times is already present in the rapidly growing field of mobile telecommunication services. In this latter context, the small mobile devices have very limited memory and hence eviction is necessary even without any prefetching considerations [8, 9, 10]. In the same context, expiration times are always being assigned to express the validity of routing as the network and its users move and change.

How does the widely practiced LRU (Least Recently Used) eviction policy adjust in the case where pages in the cache have expiration times? For example, which one of the following two pages should we prefer to evict: one containing stock market quotes which have been accessed in the very recent past but which will be invalid in a short amount of time, or one containing guidelines of the American Cancer Society that has not been accessed recently but whose information will be presumably valid for quite some time in the future? The intuition behind LRU is that our best guess for the future is that it "mirrors" the past, thus the least recently used item is also the one requested farthest

*Georgia Institute of Technology. Research supported by NSF under grant CCR-9732746. parik@cc.gatech.edu.

†AT&T Research Labs. howard@research.att.com.

‡Georgia Institute of Technology. Research supported by NSF under grant CCR-9732746. aranyak@cc.gatech.edu.

§Georgia Institute of Technology. Research supported by NSF under grant CCR-9732746 and by a Georgia Tech Edenfield Faculty Fellowship. mihail@cc.gatech.edu.

¶Georgia Institute of Technology. Research supported by NSF under grant CCR-9732746. nkv@cc.gatech.edu.

in the future. If we carry this intuition to the case of expiration times, we might think it is beneficial to keep in the cache items without recent use, if these items have very long expiration times, and are thus projected to be valid the next time they are requested. And on the other hand, we might want to evict items that were used quite recently if such items have very short expiration times and are projected to be stale the next time that they will be requested. What kind of adaptations does LRU need?

Our results suggest that, in the framework of competitive analysis of online algorithms, LRU can be carried out effectively with minimal adaptations, and without any of the above additional complicating heuristics. This can be also viewed as reaffirming current practice which does not invest additional resources towards optimizing eviction strategies. It turns out that even small additional computation at the stage of eviction decision causes large total overhead, and practitioners are very particular about keeping that overhead low.

The rest of the paper is organized as follows. From a technical point, we capture the difficulties introduced by expiration times in the paging problem (where paging is caching when all items have equal size) with a "covering" invariant; this is explained in Section 2.

In Section 3 we establish k-competitiveness for a class of deterministic marking algorithms when pages have expiration times, where hereafter k is the size of the cache. This matches the performance of marking algorithms without expiration times [5, 26]. Our algorithms use a minimal adaptation of the classical marking algorithms: they only make sure not to keep many stale copies of the same page. The proof in Section 3 is an adaptation of phases of the usual k-competitiveness proof.

In Section 4 we establish $2H_k$-competitiveness for randomized marking algorithms, matching the performance of the randomized marking algorithm in [16]. The proof in Section 4 requires counting arguments beyond the usual $2H_k$ proof for randomized marking.

As the expiration times become shorter, the competitive ratio should improve. In fact, for very short expiration times we might expect that the cache is of little use and the online algorithm should match the performance of the offline. In Section 5 we obtain upper and lower bounds within factor 2; these bounds become tight for "small" and "large" expiration times.

In Section 6 we discuss the offline problem. We obtain an optimal offline algorithm with running time $n^{O(k)}$, and a factor 3 approximation algorithm with running time polynomial in n and k. The exact complexity of the offline problem remains open (in contrast to the known flow-based polynomial time algorithm for the offline problem when there are no expiration times [22]).

We outline experimental results in Section 7.

2 Preliminaries

Consider a universe $\mathcal{U}$ of equal sized pages maintained in slow memory and a fast memory, or cache, which can hold k pages; k is typically much smaller than the total number of pages. We will maintain for convenience that cache starts with k dummy pages, which are never requested. Consider a sequence of page requests $\sigma = r_1, r_2, \ldots, r_n$, $r_i \in \mathcal{U}$. In the standard model, the requirement is that a copy of page r_i is in the cache at time i. If a copy of page r_i is not in the cache at time i then it must be fetched from the slow memory at unit cost. This is also called a *fault* or a *miss*. However, since the cache can hold at most k pages, pages must be also evicted. Thus, the question is to devise eviction policies that minimize the number of faults.

In the offline scenario where the whole request sequence is known in advance, it is well known that evicting the page whose next request is farthest in the future is an optimal strategy[2].

In the online scenario the request sequence is presented one request at a time, and the performance of the algorithm is measured by comparing it to the performance of an optimal offline algorithm. The following class of so-called *marking* algorithms have been well studied [5, 26]:

Algorithm Mark.0

```
repeat r = next request;
    if r ∈ cache, then mark r;
    if r ∉ cache, then
      if all pages are marked then
        unmark all pages;
      evict an unmarked page;
      fetch and mark r;
```

In particular, it is well known that the above class of marking algorithms are *k-competitive*, where k-competitiveness means that, the ratio of the number of faults of the algorithm to the number of faults of the optimal offline algorithm is bounded by k, for all request sequences [5, 26]. In [16], the randomized variant of Mark.0 which evicts an unmarked page chosen uniformly at random among all unmarked pages is shown to be $2H_k$ competitive, in expectation. (Here $H_k = \sum_{i=1}^{k} \frac{1}{i}$, and expectation is over the coin tosses of the online algorithm; see [5, 21, 25] for complete details).

Now suppose that when a copy of page r_i is brought from slow memory at time i, it is also assigned a *time-to-live* τ_i, the interpretation being that the particular copy

of the page is *fresh* or *valid* only until time $i + \tau_i$ (see Figure 1); a copy that is not fresh is called *stale*. The requirement now becomes that a fresh copy of page r_i is in the cache at time i, and we want to serve the request sequence incurring a small number of misses. Note that the expiration times are assigned by the slow memory, and not by the cache manager. When the cache loads a copy of a page from slow memory, then it obtains this copy together with an expiration time; the cache has no prior knowledge of expiration times.

We place the following natural restriction on the expiration times. If a copy of page p fetched from slow memory at time i is assigned time-to-live τ_i, and is thus fresh until $i + \tau_i$, then any copy of page p fetched from slow memory at times $j \in [i+1, i+\tau_i]$ must be fresh until at least time $i + \tau_i$, thus must be assigned expiration times $\tau_j : j + \tau_j \geq i + \tau_i$. This will be referred to as the *monotonicity* assumption. The monotonicity assumption expresses the intuitive fact that if the information on a certain page fetched from CNN at 9 am is determined to be valid until 12 noon, then the information on the same page fetched later at, say, 10 am cannot be determined to be valid only until 11 am. Of course, we are not imposing any restriction on expiration times for different pages. Indeed it is quite likely that, for example, stock market quotes are consistently assigned much shorter times-to-live than other pages.

Let us now introduce a convenient geometric representation of the problem which formalizes the arrows of Figure 1. For each page p requested at time i, if a copy of page p fetched from slow memory at time i is assigned time-to-live τ_i then we introduce an interval of *type* p that starts from i and ends at $i+\tau_i$. Now we say that an interval $[i, i+\tau_i]$ of type p *covers* a request for page p at time j if and only if $i \leq j \leq i+\tau_i$. Finally, for any part of the request sequence starting at $s \geq 1$ and ending at $t \leq n$, we define the *covering requirement* of page p from s to t as the minimum number of intervals of type p that cover all requests for page p between s and t. We denote the *covering requirement* of page p from s to t by $\alpha_s^t(p)$. Realize that going from time 1 to n greedily choosing intervals of type p yields a set of segments of size $\alpha_1^n(p)$ that covers all the requests for page p. The following are consequences of these definitions.

PROPOSITION 2.1. *Any algorithm faults at least* $\sum_{p \in \mathcal{U}} \alpha_1^n(p)$ *times.*

PROPOSITION 2.2. *Between times* s *and* t, *any algorithm faults at least* $\sum_{p \in \mathcal{U}} (\alpha_s^t(p) - 1)$ *times.*

For Proposition 2.2, realize that at time s, the algorithm may already have a copy of page p in cache, in which case it will need to fault only $\alpha_s^t(p) - 1$ times for page p. However, when s is 1, as in Proposition 2.1, since we begin with all dummy pages, it cannot use any of the initial contents of the cache.

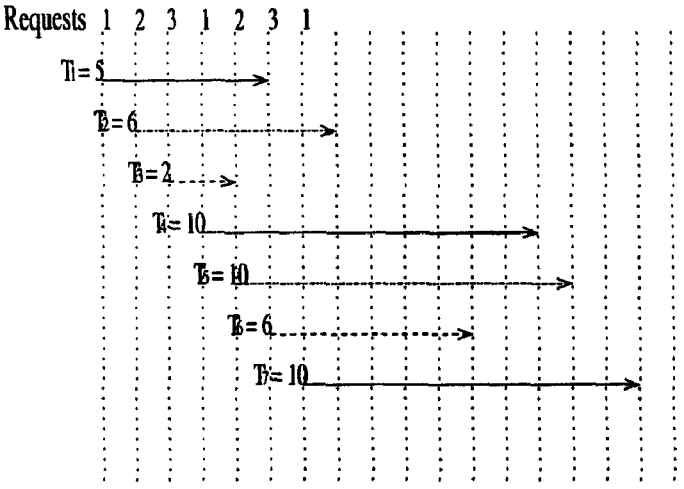

Figure 1. In the above instance, the request sequence is $1, 2, 3, 1, 2, 3, 1$ and the times to live are $5, 6, 2, 10, 10, 6, 10$. The cache is of size 2. The optimal eviction policy is as follows. At time 1, bring in page 1. At time 2, bring in page 2. At time 3, bring in page 3, evict page 1. At time 4, bring in page 1 and evict page 3. At time 5, there is a fresh copy of page 2 in the cache. At time 6, bring in page 3, evict page 2. At time 7, there is a fresh copy of page 1 in the cache. The covering requirements are $\alpha_1^7(1) = 2, \alpha_1^7(2) = 1, \alpha_1^7(3) = 2$. However $\alpha_3^7(1) = 1$.

3 Deterministic Online Marking Algorithms

In this section we define a suitable modification of marking algorithms when the pages have expiration times and a k-competitiveness proof. This proof is a careful adaptation of the usual agrument by suitably readjusting the notion of "phases".

Algorithm Mark.1

```
repeat r = next request;
  if r ∈ cache and r is fresh, then mark r;
  if r ∈ cache and r is not fresh, then
        evict r;
        fetch and mark r;
  if r ∉ cache, then
        if all pages are marked, then unmark all pages;
        evict an unmarked page;
        fetch and mark r;
```

THEOREM 3.1. *Mark.1 is* k*-competitive.*

To prove the theorem, we divide the request sequence into *phases*, where a new phase begins just before the request of the $k+1$st distinct item. The start of a phase coincides with the point when the algorithm unmarks

all the pages in its cache. Let P_i be the set (and not multiset) of pages requested in phase i. $|P_i| = k$. Suppose phase i begins at time s and ends at time t. Lemma 3.2 below follows from the fact that Mark.1 does not evict a marked page from its cache.

LEMMA 3.1. *Mark.1 will fault at most* $\sum_{p \in P_i} \alpha_s^t(p)$ *times during phase* i.

Now define *period* i to be the set of requests starting from the second request of phase i and continuing till the first request of phase $(i+1)$.

LEMMA 3.2. *Any algorithm will fault at least* $\sum_{p \in P_i} (\alpha_s^t(p) - 1) + 1$ *times during period* i.

Any algorithm begins period i with a copy of the first request of phase i in its cache. During period i, k more distinct pages are requested. Hence the algorithm has to fault on the first request of at least one of these pages. In addition to this, it will fault a further $\sum_{p \in P_i} (\alpha_s^t(p) - 1)$ times, by Proposition 2.2.
Hence the ratio of faults by MARK1 in phase i to the number of faults by the optimal algorithm in period i is

$$\frac{\sum_{p \in P_i} (\alpha_s^t(p) - 1) + k}{\sum_{p \in P_i} (\alpha_s^t(p) - 1) + 1} \leq k.$$

Now Theorem 3.1 follows.

Remark on LRU : It is easy to show that the following modification of LRU is a marking algorithm by the above definition. On a request for page p, if there is a fresh copy of p in the cache, use it. If there is a stale copy of p in the cache, replace it with a fresh copy. If there is no copy of p in the cache, evict the least recently used page and bring in a copy of p.

4 A Randomized $2H_k$ Competitive Algorithm

The randomized marking algorithm, RMA, is defined exactly like Mark.1, except that when the requested page is not in the cache (fresh or otherwise), it evicts a page chosen uniformly at random from the set of unmarked pages in the cache. However, the proof of $2H_k+3$ competitiveness requires a more careful counting argument than the standard $2H_k$ proof [16] when there are no expiration times.

Algorithm RMA

```
repeat r = next request;
  if r ∈ cache and r is fresh, then mark r;
  if r ∈ cache and r is not fresh, then
        evict r;
        fetch and mark r;
  if r ∉ cache, then
        if all pages are marked, then unmark all pages;
        evict an unmarked page
                    chosen uniformly at random
                    among all unmarked pages;
        fetch and mark r;
```

THEOREM 4.1. *The randomized marking algorithm* RMA *is* $2H_k + 3$ *competitive.*

As in the last section, let P_i be the set of pages requested in phase i. There are k distinct pages in the cache in the beginning of phase i. During the phase, k distinct pages are requested, some of which may be in the cache at the start of the phase. For phase i, let A_i be the set of pages requested in phase i which are in the cache at the start, but which expire before the first time that page is requested during the phase. Let B_i be the set of pages in the cache that are fresh till the first time they are requested in phase i. Let C_i be the set of pages requested in phase i, which are not in the cache at the start of the phase. There are $|C_i|$ pages in the cache at the end of phase $i-1$ which are not requested in phase i. Let $a_i = |A_i|, b_i = |B_i|, c_i = |C_i|$. Note that $a_i + b_i + c_i = k$.

The algorithm will fault on the first requests for pages in $A_i \cup C_i$. The number of faults on the first requests for the pages in B_i is a random variable Z_i. Note that the $|C_i|$ pages not used in phase i may be used in some future phase. Hence the only times when an algorithm evicts a fresh page is when it evicts one of these pages or a page from the set B_i. Suppose the algorithm evicts p fresh and q stale pages in the complete run, over all phases. The segments corresponding to stale pages form an independent set that does not entirely cover the request sequence. Hence, by Proposition 2.1, q is no more than the total number of faults incurred by the optimal offline algorithm. So to bound the performance, we need to just bound the number of fresh pages evicted. Let p_i be the number of fresh pages evicted by RMA during phase i. By the preceeding argument we have $p_i \leq c_i + Z_i$.

We now calculate the expected value of Z_i. Order the pages in B_i according to the time when they are first requested. We calculate the probability of a fault on the first request for the jth page in B_i. Suppose that l_j pages from C_i and m_j pages from A_i have already been requested. There are $k - (m_j + j - 1)$ pages which were in the cache at the beginning of the phase, and which have not already been requested. It can be proved by induction that the l_j pages from C_i have effectively replaced a set of size l_j uniformly at random from these pages. The probability that the jth page from B_i is in this subset is $\frac{l_j}{k-(m_j+j-1)}$.

$$
\begin{aligned}
E(Z_i) &= \sum_{j=1}^{b_i} \frac{l_j}{k-(m_j+j-1)} \\
&\leq \sum_{j=1}^{b_i} \frac{c_i}{k-(a_i+j-1)} \\
&= \frac{c_i}{c_i+b_i} + \ldots \frac{c_i}{c_i+1} \\
&= c_i(H_{b_i+c_i} - H_{c_i}) \\
&\leq c_i H_k \\
\Rightarrow E(p_i) &\leq c_i(H_k+1) \\
\Rightarrow E(p) &= \sum_i (c_i(H_k+1))
\end{aligned}
$$

Let $f(OPT)$ be the total number of times the optimal algorithm faults and let $f_i(OPT)$ be the number of faults in the ith phase. Consider phases i and $i+1$. There are $k+c_i$ distinct pages requested in these two phases. Hence there are at least c_i faults in these two phases. Hence $f_i(OPT) + f_{i+1}(OPT) \geq c_i$. So $f(OPT) \geq \sum_i c_i/2$. Hence we get

$$
\begin{aligned}
\frac{f(RMA)}{f(OPT)} &= \frac{E(p)+E(q)}{f(OPT)} \\
&= \frac{E(p)}{f(OPT)} + \frac{E(q)}{f(OPT)} \\
&\leq \frac{\sum_i c_i(H_k+1)}{\sum_i c_i/2} + 1 \\
&= 2H_k + 3
\end{aligned}
$$

5 Improved Bounds for Deterministic Online Algorithms

In this section we investigate how the competitive ratio of deterministic online algorithms depends on the maximum expiration time τ. The case when all expiration times are infinity corresponds to paging without expiration times. Here the competitive ratio is k which is tight for upper and lower bounds [26, 5]. When $\tau = 1$, competitive ratio of 1 is obvious. We would like to determine upper and lower bounds on competitive ratios for intermediate values of τ.

We show that for $\tau \leq k$ the following algorithm achieves a competitive ratio of 1. When a request for page p arrives, if there is an fresh copy of the page in the cache, use it to service the request. Otherwise if there is a stale page, evict it. If there is no stale page, then there is necessarily a page p which is valid only till the present request; evict it.

It can be seen that the above algorithm greedily covers the requests for each page. Hence the size of the cover for page p is $\alpha_1^n(p)$. So, it achieves the lower bound established in Proposition 2.1, and hence it is optimal.

In the case when $\tau > k$ we establish a lower bound which tends to k for large τ. We also show an upper bound on the competitive ratio of a suitable modification of marking algorithms in this case.

5.1 Lower Bounds on the Competitive Ratio

We prove a lower bound on the competitive ratio c_A of any deterministic online algorithm A as a function of τ. Given an expiration time τ, we construct a request sequence for $k+1$ distinct pages as follows. Assume that the caches of both the online algorithm A and the offline algorithm are filled with some dummy pages initially. We start by requesting pages $1, \ldots, k+1$. At each successive step, we request the page from $\{1, 2, ..., k+1\}$ that A does not have in the cache. Proceeding thus, we construct a sequence of length $\tau + 1$. We assign expirations times such that any page brought in at any time expires exactly at time $\tau + 1$. It is easy to see that the request sequence is monotonic and that the maximum expiration time is τ. The algorithm A faults on every request of this sequence.

We now service the request using Farthest in the Future (FF). (In Section 6, we will show that FF is not optimal in general for caching with expiration times. However, it gives us an upper bound on the number of faults by the optimal algorithm, which is what we need to show a lower bound on the competitive ratio.)

FF faults k times for the first k requests, because it also starts with dummy pages in the cache. Beyond that, FF faults just once at the start of each phase, where a phase is as defined in Section 3. There are at most $\lceil \frac{\tau-k}{k} \rceil$ phases after the first k requests, this happens when each phase is of length exactly k. Hence FF faults at most $k + \lceil \frac{\tau-k}{k} \rceil$ times until time $\tau + 1$.

After time $\tau+1$, all the pages in the caches of both A and FF will have expired, so we can repeat the same sequence all over again. Hence, $c_A \geq \frac{\tau}{k+\lceil \frac{\tau-k}{k} \rceil}$. Let $\rho = \lceil \frac{\tau}{k} \rceil$. Hence $c_A \geq \frac{\rho k}{\rho+k}$. When ρ is large compared to k, this ratio is close to k.

5.2 Upper Bounds on the Competitive Ratio

We derive upper bounds on the competitive ratio of marking algorithms as a function of the maximum expiration time. To derive a better upper bound than k, we redefine the notion of marking algorithms and phases in a way that allows us to bound the number of times a marking algorithm faults per phase. We will call this

new type of phase an *epoch*, so as to avoid confusion between the two types of phases.

Algorithm Mark.2

```
repeat
  r = next request;
  if r ∈ cache and r is fresh then mark r;
                          else
      if all pages are marked then unmark all pages;
      evict an unmarked page;
      fetch and mark r;
```

THEOREM 5.1. *Let $\rho = \lceil \frac{\tau}{k} \rceil$, where τ is the maximum expiration time. Then $c_{Mark.2} \leq min(\rho + 1, k)$.*

We analyze Mark.2 in epochs. The first epoch begins at time 1. Suppose an epoch has begun at time s. Then this epoch ends, and the next epoch begins at time t, where t is the minimum time such that $\sum_p \alpha_s^t(p) = k + 1$. It can be seen that the start of a new epoch coincides precisely with the time when the marking algorithm unmarks all the pages in the cache.

Let P_i be the set (and not multiset) of pages requested in the ith epoch. Suppose epoch i begins at time s and ends at time t. It is clear from the definition of an epoch that $\sum_{p \in P_i} \alpha_s^t(p) = k$. Though it is not necessary that any algorithm will have to fault k times, it is true that any algorithm will need at least $\alpha_s^t(p)$ distinct copies of page $p \in P_i$, and hence, totally, k distinct copies of pages in P_i to service all requests. Since Mark.2 does not evict marked pages, it is clear that it faults at most k times.

We now derive a lower bound on the number of faults for the optimal algorithm OPT. Let $\rho = \lceil \frac{\tau}{k} \rceil$. Divide the input sequence into epochs. Suppose there are N epochs, and assume, for simplicity, that the last epoch was completed. Let $f_1 \ldots f_N$ be the number of faults in each epoch and let $f(OPT) = \sum_{i=1}^{N} f_i$.

Now consider the last epoch. At least k distinct copies of pages in P_N are needed to service all the requests in epoch N. If the algorithm faults f_N times, the other $k - f_N$ pages would have been brought in during previous epochs. Also, each epoch is of length at least k. Since the expiration time is bounded by ρk, these pages must have been brought in during the epochs $n-1, n-2 \ldots n-\rho$, since all pages brought before that have expired. This implies

$$\begin{aligned} f_{N-1} + f_{N-2} \cdots + f_{N-\rho} &\geq k - f_N \\ \Rightarrow f_N + f_{N-1} \cdots + f_{N-\rho} &\geq k \end{aligned} \tag{5.1}$$

Similarly we get

$$f_{N-\rho-1} + f_{N-\rho-2} \cdots + f_{N-2\rho-1} \geq k \tag{5.2}$$

We can continue in this fashion in groups of $\rho + 1$, until the last group which may be of size less than $\rho + 1$. But since $f_1 = k$, the sum of faults in the last group is also at least k.

Adding all the equations we get:

$$f(OPT) \geq \left\lceil \frac{N}{\rho+1} \right\rceil k \geq \frac{Nk}{\rho+1} \tag{5.3}$$

Also, since there are N epochs and Mark.2 faults at most k times per epoch, the total number of times it faults, $f(Mark.2) \leq Nk$. Hence, $c_{Mark.2} \leq \frac{Nk}{(Nk)/(\rho+1)} \leq \rho + 1$. By analysis similar to that used for Mark.1 in Section 3, we can show that $c_{Mark.2} \leq k$. Hence we get $c_{Mark.2} \leq min(\rho+1, k)$, proving Theorem 5.1.

6 The Offline Problem

The following simple modification to farthest in the future may seem to be a good candidate for the optimal offline algorithm in our model. Upon a cache miss, evict a page in the cache which is either stale at the current time, or will become stale before the next time it is requested. If no such page is found then evict a page whose next request is farthest in the future, among all pages in the cache.

The request sequence in Figure 1 shows, however, that this algorithm is not optimal. At time 3, when page 3 is requested, we need to evict either page 1 or page 2. Though page 2 is requested farther in the future, the optimal strategy is to evict page 1.

The problem of finding the optimal offline algorithm can be easily modeled as a shortest path problem. The vertices of the graph correspond to all possible configurations of the cache. At time t, there are at most $\binom{t}{k}$ possible configurations. Hence the total number of nodes is bounded by n^{k+1}. A vertex representing a configuration at time t is connected to those vertices representing configurations at time $t + 1$ which can be reached. The edge is assigned a weight of 1 or 0 depending on whether or not the algorithm must fault in order to make the transition. We add a source connected to all configurations at time 1 and a sink connected from all final configurations by edges of cost 0. The problem reduces to finding a shortest path from the source to the sink.

The running time of this algorithm is $n^{O(k)}$. We have not found an algorithm polynomial in both n and k. The question of whether it is NP-complete for arbitrary k is open. We present a 3-factor approximation algorithm for the problem.

A 3-factor Approximation Algorithm

We follow the terminology used in Section 4. The algorithm proceeds in phases, much like RMA, except that at the start of phase i, since it knows the entire request sequence, it can identify the sets A_i, B_i and C_i. Our algorithm evicts the A_is and then the c_i pages which are not requested in this phase. These c_i pages are the only pages that can possibly be fresh when evicted. Note that the number of new requests in this phase is $k - a_i - b_i = c_i$. So we do not need to evict the B_is. Suppose that p fresh and q stale pages are evicted. $p \leq f(OPT)$, while $q = \sum_i c - i$, hence $q \leq 2 * f(OPT)$. So $p + q \leq 3 * f(OPT)$.

The modified version of farthest in the future is a 3-factor algorithm, since if it evicts a fresh page, that page is not requested again in the same phase.

7 Outline of Experiments

We have implemented the following algorithms: (a) LRU-A which does not take in to account any expiration times, but makes sure not to keep many stale copies of the same page (as in Section 3). (b) LRU-B which simply prefers to evict stale pages before evicting any fresh page. (c) LFU, least frequently used. (d) "Mirror" mentioned in the introduction: for each page in the cache let t_{past} be the time that has elapsed since it was last used, and let t_{future} be the future time for which it will be fresh. The algorithm evicts the page which minimizes the ratio t_{future}/t_{past}. (e) "Log-Mirror" which evicts the page minimizing the ratio $\log(t_{future})/t_{past})$.

For real data, we have used traces from [15] and [24]. We have also generated synthetic data using random walks (to simulate locality of reference) and sampling from varying skewed distributions.

We have found that the simplest implementation of LRU-A performs generally as well as any more sophisticated scheme. In very few instances Log-Mirror was better, which leaves another open issue to explore.

Further Work

There is a gap between the upper and lower bounds on the competitive ratio in Theorem 5.1. Perhaps tighter bounds are possible. Also, the algorithms here do not make use of the time to live. It may be possible to get a better competitive ratio using this information. The status of the optimal offline problem is open.

Acknowledgment: We thank Elias Koutsoupias and Vijay Vazirani for useful discussions.

References

[1] Susanne Albers, Generalized Connection Caching, *SPAA* 2000.

[2] L.A. Belady. A study of replacement algorithms for virtual storage. *IBM Systems Journal*, pp 5:78-101, 1966.

[3] T. Bernes-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol. HTTP 1.0. RFC 1945. MIT/LCS, May 1996. http://ds.internic.net/rfc/rfc1945.txt

[4] M.A. Blaze. *Caching in Large-Scale Distributed File Systems*, Ph.D. Thesis, Princeton University, Jan. 1993.

[5] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.

[6] P. Cao and S. Irani. Cost aware WWW proxy caching algorithms, Technical Report 1343, Dept. of Computer Science, U. Wisconsin, Madison, 1997, also in *2nd Web Caching Workshop*, Boulder, Colorado, 1997.

[7] P. Cao and C. Liu. Maintaining Strong Cache Consistency in the Wold-Wide Web, *Proc. ICDCS97*, pp 12-21, May 1997.

[8] Tamra Carpenter, Robert L. Carter, Munir Cochinwala, Martin Eiger, Client-Server Caching with Expiration Timestamps. *Distributed and Parallel Databases*, 10(1): 5-22 (2001)

[9] Tamra Carpenter, Robert L. Carter, Munir Cochinwala, Martin Eiger, Caching for Mobile Communication. *ADBIS-DASFAA* 2000: 37-50.

[10] Munir Cochinwala, Database Performance for Next Generation Telecommunications. *ICDE* 2001: 295-298

[11] E. Cohen and H. Kaplan, Prefetching the means for document transfer: A new approach for reducing Web latency. IEEE INFOCOM'00.

[12] E. Cohen and H. Kaplan, Proactive caching of DNS records: Addressing a performance bottleneck. 2001 Symposium on Applications and the Internet (SAINT). IEEE, 2001.

[13] E. Cohen and H. Kaplan, Refreshment policies for Web content caches. IEEE INFOCOM'01.

[14] E. Cohen and H. Kaplan, The age penalty and its effect on cache performance. USENIX Symposium on Internet Technologies and Systems (USITS). 2001.

[15] A Distributed Testbed for National Information Provisioning, http://www.ircache.net.

[16] 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.

[17] M. Franklin. *Client Data Caching: A Foundation for High Performance Object Database Systems*, Kluwer Academic Publishers, 1993.

[18] J. Gwertzman and M. Seltzer, World Wide Web Cache Consistency, *Proc. 1996 USENIX Technical Conference*, San Diego, CA, Jan 1996.

[19] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, 1990.

[20] S. Irani. Page replacement with multi-size pages and applications to web caching, *Proc. 29th ACM Symposium on Theory of Computing* pp 701-710, 1997.

[21] S. Irani and A.R. Karlin. Online Computation, Approximation Algorithms for NP-Hard Problems, edited by D.S. Hochbaum, PWS Publishing Company, 1997.

[22] H. Karloff and S. Sundarajan, A Polynomial Time Algorithm for the Offline k-Server Problem.

[23] Y. Kotidis and N. Roussopoulos, DynaMat: A Dynamic View Management System for Data Warehouses, *Best Paper Award, SIGMOD* 1999.

[24] R. Liston and E. Zegura, *The Design and Evaluation of a Method for Measuring Web Caching Performance*, In Proceedings of the 6th International Web Caching Conference, May 2001.

[25] R.Motwani and P. Raghavan. *Randomized Algorithms*, Cambridge University Press, 1995.

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

[27] Squid Internet Object Cache. http://suid.nlanr.net/Squid

[28] Jia Wang, A Survey of Web Caching Schemes for the Internet, *ACM Computer Communication Review* (CCR), Vol. 29, No. 5, October 1999.

[29] D. Wessels. Intelligent Caching for WWW Objects, *Proc. INET-95*, 1995.

[30] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, H. Levy, On the scale and performance of cooperative Web proxy caching. *Proc. of the 17th ACM Symposium on Operating Systems Principles* SOSP '99), December 1999.