Caching with Expiration Times for Internet Applications*

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

April 23, 2005

Abstract

Caching data together with expiration times beyond which the data is no longer valid is a standard method for promoting information coherence in distributed systems, including the Internet, the World Wide Web (WWW) and Peer-to-Peer (P2P) networks. We use the framework of competitive analysis of online algorithms, and study upper and lower bounds for page eviction strategies in the case where data has expiration times. We show that minimal adaptations of marking algorithms achieve performance similar to the well studied case of caching without the expiration time constraint. Marking algorithms include the widely used Least Recently Used (LRU) eviction policy. In practice, when data have expiration times, the LRU eviction policy is used widely, often without any consideration of expiration times. Our results explain and justify this standard practice.

Keywords: Caching, Internet, WWW, Online Algorithms, Expiration Times.

*Research supported by NSF under grant CCR-9732746 and by a Georgia Tech Edenfield Faculty Fellowship.

†AT&T Research Labs, NJ. Email: karloff@research.att.com

‡Georgia Institute of Technology, Atlanta, GA 30332. Email: {parik,aranyak,mihail,nkv}@cc.gatech.edu

1 Introduction

Caching data together with expiration times beyond which the data is no longer valid is a standard mechanism for promoting information coherence in distributed systems. It has been considered in the context of distributed memory in computer architecture [16], distributed memory in databases [12], and, more recently, in large database or datawarehouse maintenance [21, 15, 25]. Most important, caching data together with expiration times is becoming the method of choice in Internet and WWW applications [2, 3, 6, 9, 10, 14, 24, 26]. For example, a WWW server, proxy, or client (browser) may cache pages together with an estimate of the time before the information in the page becomes stale -also known as *time-to-live* (TTL), expiration times are used to promote consistency in Domain Name Server (DNS) resolution, and even in advanced telephony expiration times are used in the context of translations of 800-numbers, mobile and personal communications services. The most appealing characteristic of this expiration time assignment method for data consistency is simplicity of implementation.

In the context of the Internet, an important consideration in caching seems to be the very decision as to what are appropriate expiration time assignments [2, 9, 10, 14, 24]. In the context of content distribution and DNS resolution a solid body of studies focus on issues concerning latency and size [5, 18], as well as pre-emptive strategies, like prefetching and its ramifications [7, 8, 9, 10]. In this paper we focus on cache eviction policies.

How does the widely practiced LRU eviction policy adjust in the case where pages in the cache have expiration times? If requests are generated by a fixed probability distribution, then it can be verified that the most frequently used items, normalized by the length of their expiration times, should be kept in the cache. Is such a normalization necessary when the request sequence does not follow any particular pattern? 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 in the Future (FF). If we carry this intuition to the case of expiration times, we might think it 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.

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 heuristics. This can be also viewed as reaffirming current practice which does not invest valuable resources towards optimizing eviction strategies. It is therefore reassuring to know that, at least from the point of view of competitive analysis, such additional resources are not necessary.

The problem of caching with expiration times was first studied by Kimbrel [20]. He obtained deterministic algorithms for the general case where pages have expiration times, varying sizes as well as varying costs to fetch each page. He shows that even in this scenario, a natural extension of LRU and some other algorithms achieve a competitive ratio of k where hereafter k denotes the size of the cache. Thus our two papers come to the same conclusion. For deterministic algorithms, the result in [20] is stronger, since it refers to a more general model. The proof of competitiveness in [20] uses a technical potential function argument, but our result is relatively simple.

Our result for deterministic online algorithms (Theorem 9) concerns only the case where all the pages have the same size. However, for this case, our proof is much simpler. For randomized algorithms, we show that a simple randomized marking algorithm is $(2H_k)$ competitive. We show

that if the expiration times are all bounded by $\tau_{max} < k^2 - k$, then the lower bound of k is no longer valid. We present a simple class of algorithms that achieve a ratio of $\lceil \frac{\tau_{max}}{k} \rceil + 1$. We give a lower bound on the competitive ratio as a function of τ_{max} that tends to k for large values of τ_{max}. We also study the complexity of the offline problem. Without expiration times, the strategy of evicting the page whose next request is farther in the future (FF) is the optimal offline strategy [1]. We show that a natural extension of FF is no longer optimal with expiration times (A similar observation is made in [20]). We show that a natural extension of FF achieves an approximation factor of 3. We give an optimal offline algorithm that runs in time $n^{O(k)}$ where n is the length of the request sequence. The exact complexity of the offline problem remains open.

In Section 2, we briefly review some classical results about caching in the standard model. We then formally define the problem when the pages have expiration times. We introduce the notion of a *covering requirement* for each page which captures the technical difficulties introduced by expiration times. We analyze deterministic algorithms in Section 3 and randomized algorithms in Section 4. In Section 5 we give better upper and lower bounds on the competitive ratio as a function of the maximum expiration time τ_{max}. We study the offline problem in Section 6.

A preliminary version of our paper appeared in [13].

2 Preliminaries

In this section we briefly review the standard caching problem and state some classical results. We give a formal statement of the problem of caching when the pages are assigned expiration times beyond which they are no longer valid. We define the notion of a covering requirement which is used in our analysis.

2.1 The Classical Caching Problem

Consider a universe 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 K of pages in the universe. We will assume, for convenience, that the cache starts with k dummy pages, which are never requested. Consider a sequence of n page requests $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_n$. In the standard model, the requirement is that a copy of page σ_i is in the cache at time i, otherwise 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. 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 (FF) is an optimal eviction policy [1]. In the online scenario, the request sequence is presented one request at a time. Thus, when the cache is full and a page must be fetched from slow memory, then the decision of which page to evict must be made without knowing the rest of the request sequence. Naturally, this lack of information causes online algorithms to fault more frequently than offline algorithms.

Competitive analysis measures the performance of an online algorithm by comparing its number of faults on a sequence of requests to the number of faults of an optimal offline algorithm for the same request sequence, and taking the worst possible ratio over all sequences. For an algorithm A, let $f_A(\sigma)$ denote the number of faults incurred on request sequence σ. Let $f_{OPT}(\sigma)$ denote the number of faults incurred by the optimal offline algorithm.

Definition 1 *The online algorithm A is c-competitive if there is a constant b such that for every request sequence σ,*

$$f_A(\sigma) \leq c \cdot f_{OPT}(\sigma) + b$$

If the algorithm A is randomized, we replace $f_A(\sigma)$ by $E(f_A(\sigma))$ where the expectation is over the coin-tosses of A.

The following class of so-called *marking* algorithms have been well studied and are known to be k-competitive [17]. Sleator and Tarjan show a lower bound of k on the competitive ratio of any deterministic online algorithm [23].

Algorithm 1 MARK

Initially the cache contains k dummy pages, all unmarked;
repeat
  $r :=$ next request;
  if $r \in$ cache, then mark r;
  if $r \notin$ cache, then begin
    if all pages are marked then unmark all pages;
    evict an (arbitrary) unmarked page;
    fetch and mark r;
  end;

In [11], the randomized variant of MARK, which evicts an unmarked page chosen uniformly at random among all unmarked pages (instead of an arbitrary one), is shown to be $(2H_k)$-competitive, where $H_k = \sum_{i=1}^{k} \frac{1}{i}$. In fact this bound is tight. Detailed statements and proofs of these results can be found in the survey by Irani [17].

2.2 Caching with Expiration Times

We study the problem of caching when the pages are assigned expiration times. When page p is brought from main memory at time i, it is also assigned a *time-to-live* (TTL) $\tau_p(i)$. This copy of the page is *fresh* only until time $i + \tau_p(i)$. After time $i + \tau_p(i)$ the copy of the page becomes *stale* and a new copy will need to be fetched to serve requests. Thus an input instance consists of a request sequence σ where $\sigma_i \in \{1, \cdots, n\}$ and a set of expiration times $\tau_p(i)$ which specify the expiration times for the pages. Given the request sequence σ, the requirement is that a fresh copy of page σ_i is in the cache at time i. We want to serve the request sequence causing as few faults as possible. In the classical case where there are no expiration times, one may assume without loss of generality that succesive requests are for different pages. However, in our model with expiration times, succesive requests for the same page are meaningful.

We will assume that the expiration times satisfy the following requirement which we call the **monotonicity assumption:** *Copies of page p that are fetched later expire later. In other words, if $i < j$ then $i + \tau_p(i) \leq j + \tau_p(j)$.*

This is a natural assumption since it says that if a page fetched from CNN headline news at 9 am will be fresh until noon, then it cannot be the case that the same page fetched at 10 am

is fresh only until 11 am. Note that we are not imposing any restriction on requests for different pages. Indeed it is quite likely that, for example, stock market quotes are consistently assigned much shorter times-to-live than headline news.

The monotonicity assumption also allows us to restrict ourselves to *lazy* algorithms. We say that an algorithm is lazy if it fetches a page p at time i only if $\sigma_i = p$ and we do not have a fresh copy of p in the cache. Since the monotonicity assumption guarantees that a page which is fetched later will stay fresh longer, we can bring pages into the cache only when absolutely necessary. Without the monotonicity assumption it might happen that for some page p, at a certain time i, $\tau_p(i)$ is very large while at all other times, it is small. The best strategy might to bring p into the cache at time i, even if there is a fresh copy already in the cache or $\sigma_i \neq p$.

The monotonicity requirement allows us to specify $\tau_p(i)$ only for the page p where $\sigma_i = p$ since this is the only page we might bring in at time i. Thus we may assume that the input consists a pair of sequences (σ, τ), where $\sigma(i) = p$ is the i^{th} request and $\tau(i) = \tau_p(i)$ is the TTL if a copy of p is fetched at time i.

2.3 The Covering Requirement

We introduce a convenient graphical representation of the problem. Using this we show that expiration times impose a certain *covering requirement* for every page p, *i.e.* a minimum number of faults on requests to p for any algorithm whether online or offline. This does not hold in the classical model. All the bounds in this section are independent of the cache size, they are purely a result of the expiration times.

For each page p and each time i where $\sigma(i) = p$, we introduce an interval $L_p(i)$ of *type* p that starts at i and ends at $i + \tau_p(i)$. The interval $L_p(i)$ can *cover* all requests for p at time j where $j \in [i, i + \tau_p(i)]$. A set S of intervals covers all requests for p if every request is covered by an interval in S. We can cover all requests to p by choosing intervals greedily. Pick the smallest i such that $\sigma(i) = p$ and i is not already covered by S. Add $L_p(i)$ to S. We denote the size of this greedy cover by $\alpha(p)$.

Lemma 1 *Any set of intervals that covers all requests for p has size at least $\alpha(p)$.*

Proof: Let S denote the greedily constructed cover above. Let the starting points of the intervals in S be $\{i_1, i_2, \cdots i_{\alpha(p)}\}$. Let T be some other cover of size $\alpha'(p)$ consisting of intervals starting at points $\{j_1, j_2, \cdots j_{\alpha'(p)}\}$. Clearly $i_1 = j_1$ since this is the only interval that can cover the request at time i_1. The interval $L_p(i_1)$ covers all requests for p till i_2. Assume that in T, i_2 is covered by $L_p(j)$. We must have $j \neq j_1$ and $j \leq i_2$. By the monotonicity assumption, we can replace j with i_2 since this will only cover more requests. Repeating this argument, we can show that $\alpha'(p) \geq \alpha(p)$. □

We say that a set of intervals of type p is independent if the intervals do not overlap.

Lemma 2 *A set of independent intervals of type p can have size at most $\alpha(p)$.*

Proof: The greedy cover S constructed above is independent, and it has size $\alpha(p)$. Let T be an independent set of size t consisting of intervals starting at points $\{j_1, j_2, \cdots, j_t\}$. Clearly $j_1 \geq i_1$ since i_1 is the first request for page p. By monotonicity, we can replace $L_p(j_1)$ by $L_p(i_1)$ and still have an independent. By repeating this argument, we can replace T by a subset of S of size t, hence $t \leq \alpha(p)$. □

It is not hard to see that the greedy cover S is the unique cover which is also independent.

Corollary 3 *Any algorithm A (online or offline) must fault at least $\alpha(p)$ times on requests to page p.*

Proof: We show that any algorithm must construct a cover for requests to p. A request for p either results in a page fault or it is served using a copy of p brought in previously. Consider the set $F(p)$ of times when A faults on a request to p and brings a new copy of p into the cache. Since other requests for page p result in a cache hit, the set of intervals $\{L_p(i)|i \in F(p)\}$ covers all the requests for p. By Lemma 1, $|F(p)| \geq \alpha(p)$. □

A similar covering requirement holds for requests to p between time s and t. For $s \leq t$, denote the subsequences of σ and τ that start at s and end at t by σ_s^t and τ_s^t. Let $r_1, \cdots, r_\ell$ denote the requests to p in this interval. We can construct a greedy cover S_s^t by picking $L_p(r_1)$, then repeatedly picking the smallest r_i that is not covered by the intervals chosen so far. Denote the size of this cover by $\alpha_s^t(p)$. Note that in general, a cover for the requests in σ_s^t can contain intervals that start before s.

Lemma 4 *Any set of intervals that covers all requests for p in σ_s^t has size at least $\alpha_s^t(p)$. Further, at least $\alpha_s^t - 1$ intervals must start between s and t.*

Proof: The first statement is proved by an argument similar to Lemma 1. We will prove the second statement. Assume that T is a cover for the requests to p in σ_s^t containing at most $\alpha_s^t - 2$ intervals starting between s and t. Let $L_p(i)$ be an interval starting at $i < s$. Since $r_1 \geq s > i$, by monotonicity the interval $L_p(r_1)$ covers all the requests to p in σ_s^t that are covered by $L_p(i)$. Thus removing all the intervals from T that start before s and replacing them by $L_p(r_1)$ still gives a cover for the requests to p in σ_s^t. This cover has size at most $\alpha_s^t - 1$ which is a contradiction. □

Corollary 5 *Any algorithm A (online or offline) must fault at least $\alpha_s^t(p) - 1$ times on requests to page p between s and t. If the cache does not contain a copy of p at time s, it must fault $\alpha_s^t(p)$ times.*

Proof: Consider the set $F_s^t(p)$ of pages used by the algorithm to serve requests to p in σ_s^t. The intervals starting at these points cover the requests to p in σ_s^t. By Lemma 4 at least $\alpha_s^t(p) - 1$ of them start between s and t. The starting points correspond to faults on requests to page p between s and t.

If the cache does not have a copy of p at time t, all the intervals start between s and t, so the algorithm faults at least $\alpha_s^t(p)$ times. □

Consider the following instance of the problem where $\sigma = \{1, 2, 3, 1, 2, 3, 1\}$ and the times to live are $\tau = \{5, 6, 2, 10, 10, 6, 10\}$. The cache is of size $k = 2$.

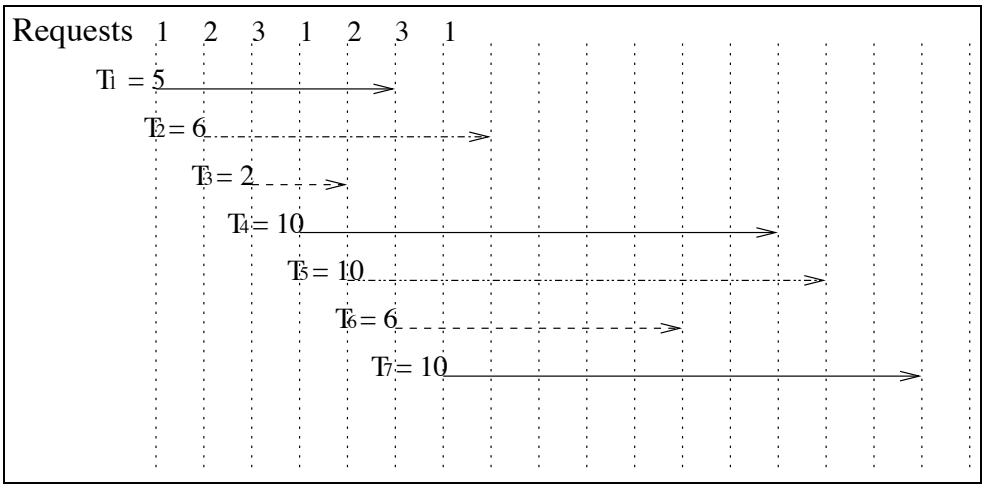

Figure 1

The optimal eviction policy is as follows. At time 1, fetch in page 1. At time 2, fetch page 2. At time 3, fetch page 3 and evict page 1. At time 4 fetch page 1 and evict page 3. At time 5, there is a fresh copy of page 2 in the cache. At time 6, fetch page 3 and evict page 2. At time 7, there is a fresh copy of page 1 in the cache. To see that this strategy is optimal, observe that it gives a total of 5 faults, and $\alpha(1) = 2, \alpha(2) = 1, \alpha(3) = 2$.

We conclude this section with an illustration that the problem with expiration times significantly different from the classical caching problem. Essentially the same observation was made in [20]. It is well known that for the offline version of the classical problem, Farthest in the Future (FF) is an optimal eviction policy [1]. In our model, the following simple modification to Farthest in the Future may seem to be a good candidate for the optimal offline algorithm.

Algorithm 2 FF2

r := next request;
If $r \notin$ cache;
  Evict a page which is stale already, or will be stale before the next request for it;
  Else evict the page whose next request is farthest in the future.

Theorem 6 *Algorithm FF2 is not an optimal offline algorithm.*

Proof: The request sequence in Figure 1 shows that FF2 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. On can see that $f_{FF2}(\sigma) = 6$, whereas $f_{OPT}(\sigma) = 5$. □

3 Deterministic Online Marking Algorithms

In this section we show that a simple modification of MARK achieves competitive ratio of k, for the generalized problem of caching with expiration times.

Algorithm 3 MARK1

initially the cache has k dummy pages, all unmarked;
repeat
  r = next request;
  if $r \in$ cache and r is fresh, then mark r;
  if $r \in$ cache and r is stale, then begin
      evict r;
      fetch and mark r;
  end;
  if $r \notin$ cache, then begin
      if all pages are marked, then unmark all pages;
      evict an (arbitrary) unmarked page;
      fetch and mark r;
  end;

To prove that MARK1 is k-competitive, we argue in *phases*. The request sequence is divided into phases, with the first phase starting at time 1. A phase ends just before the request for the $(k+1)^{st}$ distinct page. The start of a new phase coincides with the point when the algorithm unmarks all the pages in its cache. Suppose that phase i begins at time s and ends at time t. Let $P_i = \{p_1, \cdots, p_k\}$ be the set of distinct pages requested in phase i.

Lemma 7 *MARK1 will fault at most* $\sum_{j=1}^{k} \alpha_s^t(p_j)$ *times during phase* i.

Proof: Let p be a fixed page. At the start of phase i, if there isn't a fresh copy of p in the cache, then MARK1 constructs a greedy cover $S_s^t(p)$ for the requests to p in phase i. By Lemma 4, it faults $\alpha_s^t(p)$ times. Now assume that there is fresh copy of p at the start of phase i, which expires at time $s' \geq s$. If $s \geq s' < t$, MARK1 will greedily cover the requests to p in $\sigma_{s'}^t$ at a cost of $\alpha_{s'}^t(p)$. It is easy to show that $\alpha_{s'}^t(p) \leq \alpha_s^t(p)$. In the case when $s' \geq t$ MARK1 will not fault on page p in phase i. Let $f_{MARK1}(i)$ denote the number of faults of MARK1 in phase i. Then $f_{MARK1}(i) \geq \sum_{j=1}^{k} \alpha_s^t(p_j)$. □

Following [17], we define segment i to be the sequence starting at time $s+1$ and ending at time $t+1$. Let p_{k+1} be the page requested at time $t+1$.

Lemma 8 *Any algorithm will fault at least* $1 + \sum_{j=1}^{k} (\alpha_s^t(p_j) - 1)$ *times during segment* i.

Proof: The pages that could be requested in segment i are $p_1, \cdots, p_{k+1}$. The number of faults on p_1 during segment i are minimized if a fresh copy of p_1 is brought into the cache at time s. In this case, constructing a greedy cover for the remaining requests requires $\alpha_s^t(p_1) - 1$ faults. This shows that any algorithm must fault at least $\alpha_s^t(p_1) - 1$ times on request to page p_1 in segment i.

For $p_2, \cdots, p_{k+1}$ we bound the number of faults using Corollary 5. At time $k+1$, the cache contains a copy of p_1, so by the pigeonhole principle it does not contain a copy of p_ℓ for some ℓ between 2 and $k+1$. The algorithm will fault $\alpha_{s+1}^{t+1}(p_\ell)$ times on requests to p_ℓ. For $j \neq \ell$, the algorithm $\alpha_{s+1}^{t+1}(p_j) - 1$ times on page p_j during segment i. Both these statements follow from Corollary 5.

Note that for $2 \leq j \leq k$, $\alpha_{s+1}^{t+1}(p_j) = \alpha_s^t(p_j)$ and $\alpha_{s+1}^{t+1}(p_{k+1}) = 1$. Let $f_{OPT}(i)$ denote the number of faults in segment i. We have

$$\begin{aligned} f_{OPT}(i) &\geq \alpha_s^t(p_1) - 1 + \sum_{\substack{j=2 \\ j \neq \ell}}^{k+1} (\alpha_{s+1}^{t+1}(p_j) - 1) + \alpha_{s+1}^{t+1}(p_\ell) \\ &\geq \sum_{j=1}^{k} (\alpha_s^t(p_j) - 1) + 1 \qquad \text{Since } \alpha_{s+1}^{t+1}(p_{k+1}) = 1 \end{aligned}$$

□

Theorem 9 *MARK1 is k-competitive.*

Proof: We compare the faults incurred by MARK1 in phase i to those incurred by OPT in segment i. By Lemmas 7 and 8,

$$\begin{aligned} \frac{f_{MARK1}(\sigma)}{f_{OPT}(\sigma)} &\leq \frac{\sum_i f_{MARK1}(i)}{\sum_i f_{OPT}(i)} \\ &\leq \max_i \frac{f_{MARK1}(i)}{f_{OPT}(i)} \\ &\leq \frac{\sum_{j=1}^{k} \alpha_s^t(p_j)}{\sum_{j=1}^{k} (\alpha_s^t(p_j) - 1) + 1} \\ &\leq \frac{\sum_{j=1}^{k} (\alpha_s^t(p_j) - 1) + k}{\sum_{j=1}^{k} (\alpha_s^t(p_j) - 1) + 1} \\ &\leq k \end{aligned}$$

□

LRU with Expiration Times

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, then use it**.** If there is a stale copy of p in the cache, then evict the stale copy and replace it with a fresh copy. If there is no copy of p in the cache, then evict the least recently used page and fetch a copy of p.

Notice that the above algorithm does not take into account expiration times. Indeed, the algorithm may evict a fresh page (if this was the least recently used page), and keep a stale page (if this page happened to have been used recently). And yet, at least according to the competitive ratio performance measure, the algorithm achieves optimal competitiveness.

4 A Randomized $(2H_k)$-Competitive Algorithm

In this section we study a randomized eviction policy, for the case where pages have expiration times. We introduce the randomized marking algorithm RMA, defined exactly like MARK1, except that,

on being required to evict a page, the algorithm evicts a page chosen uniformly at random from the set of unmarked pages in the cache. We show that RMA achieves competitive ratio of $2H_k$.

RMA is a simple adaptation of the randomized marking algorithm of [11], where pages are not assigned expiration times. This latter algorithm was shown to achieve competitive ratio of $2H_k$ [11]. Therefore we come to the conclusion that, like deterministic marking algorithms studied in the previous section, the standard randomized eviction policies achieve good performance with minimal adaptations and, in particular, essentially without consideration of expiration times.

Theorem 10 *The randomized marking algorithm RMA is* $(2H_k)$*-competitive.*

Proof: As in the previous section, phase i starts at time s and ends at time t. The set of pages requested in phase i is denoted by P_i. At the beginning of phase i, there are k distinct pages in the cache which were brought in before the phase started. During phase i, k distinct pages are requested, some of which may be in the cache at the beginning of phase i. For phase i, let A_i be the set of pages which are in the cache at the beginning of the phase, but which will be stale the first time that they will be requested during the phase. Let B_i be the set of pages in the cache at the beginning of the phase which will be fresh the first time they will be requested in the phase. Let C_i be the set of pages requested in phase i, which are not in the cache at the beginning of phase i. Therefore, there were $|C_i|$ pages in the cache at the end of phase $i-1$ which were not requested during phase i. Let $a_i = |A_i|$, $b_i = |B_i|$ and let $c_i = |C_i|$. Notice that $a_i + b_i + c_i = k$.

RMA will certainly 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. Let us first calculate the expected value of Z_i. To do so, 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 in C_i and m_j pages in 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 seen (for example, inductively) that the l_j pages from C_i replace a set of size l_j chosen uniformly at random from these pages. The probability that the jth page from B_i is in this subset is $\frac{l_j}{k-(m_j+j-1)}$. We thus have

$$\begin{aligned}
E(Z_i) &= \sum_{j=1}^{b_i} \frac{l_j}{k-(m_j+j-1)} \\
&\leq \sum_{j=1}^{b_i} \frac{c_i}{k-(a_i+j-1)} \\
&= \sum_{j=1}^{b_1} \frac{c_i}{c_i+b_i-j+1} \\
&= \frac{c_i}{c_i+b_i} + \ldots \frac{c_i}{c_i+1} \\
&= c_i(H_{b_i+c_i} - H_{c_i}) \\
&\leq c_i H_k \qquad (1)
\end{aligned}$$

Let $f_{OPT}(i)$ be the number of faults of this optimal algorithm in the i^{th} phase. Consider phases i and $i+1$. There are $k + c_i$ distinct pages requested in these two phases. Hence the optimal

algorithm faults at least c_i times in these two phases. Hence,

$$\begin{aligned} c_i &\leq f_{OPT}(i) + f_{OPT}(i+1) \\ \Rightarrow \sum_i c_i &\leq 2f_{OPT}(\sigma) \end{aligned} \tag{2}$$

Let x be the total number of fresh pages evicted by RMA over all phases.

$$\begin{aligned} x &\leq \sum_i (c_i + Z_i) \\ \Rightarrow E[x] &\leq \sum_i c_i + \sum_i E[Z_i] \\ &\leq (1+H_k)\sum_i c_i \qquad \text{By (1)} \\ &\leq 2(1+H_k) f_{OPT}(\sigma) \qquad \text{By (2)} \end{aligned} \tag{3}$$

Let y be the total number of stale pages evicted by RMA over all phases. Let y_p denote the number of stale copies of page p. The intervals corresponding to y_p form an independent set, hence by Lemma 2 $y_p \leq \alpha(p)$. Note that both y and y_p are random variables. Hence

$$E[y] \;=\; \sum_p E[y_p] \;\leq\; \sum_p \alpha(p) \;\leq\; f_{OPT}(\sigma) \tag{4}$$

The total number of faults equals the total number of pages evicted. Hence

$$\begin{aligned} E[f_{RMA}(\sigma)] &= E[x] + E[y] \\ &\leq (2H_k + 3) f_{OPT}(\sigma) \end{aligned} \tag{5}$$

□

5 Improved Bounds for Deterministic Online Algorithms

In this section we investigate how the competitive ratio of deterministic online algorithms depends on the maximum time-to-live $\tau_{\max} = \max_{p,i}\{\tau_p(i)\}$. The case when all times-to-live are infinity corresponds to caching without expiration times, for which the well known tight upper and lower bounds of k apply [23, 4]. When $\tau_{\max} = 0$, it is obvious that the competitive ratio is necessarily 1, since every requested page has to be fetched at every time step. We would like to determine upper and lower bounds on competitive ratios for intermediate values of $\tau_{\max}$.

Theorem 11 *For $\tau_{\max} \leq k$, there is a deterministic algorithm with competitive ratio 1.*

Proof: Since $\tau_{\max} \leq k$, the only pages which could be fresh at time i are the pages that are brought in during the interval $[i-k, i-1]$. There are at most k of them so we can keep all these pages in the cache.

The algorithm is the obvious greedy algorithm. When a request for page p arrives, if there is a fault, evict a page which will be stale at time $i+1$. Such a page always exists. It is easy to show that this algorithm produces a greedy cover for each page p so it is optimal by Corollary 3. □

Any algorithm that evicts only stale pages is optimal. Since LRU will not evict a page which is used in the last $k-1$ requests, it will never evict a fresh page, hence it is optimal.

For $\tau_{\max} > k$ we establish a lower bound which tends to k, for large $\tau_{\max}$. We also give an upper bound for a simple modification of the basic marking algorithm.

Theorem 12 *When $\tau_{\max} > k$, for any deterministic online algorithm the competitive ratio is at least $\frac{\tau_{max}}{k+\lceil\frac{\tau_{\max}-k}{k}\rceil}$.*

Proof: Let A be a deterministic online algorithm. We construct a request sequence σ for $k+1$ distinct pages of length $\tau_{\max}+1$ as follows. At each time $i \leq \tau_{\max}+1$, we request a page in $1, \cdots, k+1$ that A does not have in the cache. We set $\tau(i) = \tau_{\max}+1-i$, so that all pages expire at time $\tau_{\max}+1$. From time $\tau_{\max}+2$ to $2\tau_{\max}+2$ we construct a similar sequence of requests for a different set of k pages and so on. Clearly the expiration times are bounded by $\tau_{\max}$ and they are monotone. The algorithm A faults on every request in this sequence.

We now service the request using the offline algorithm Farthest in the Future (FF). (FF is in fact the optimal offline strategy for the above request sequence, but we will not need this fact.) FF faults k times for the first k requests. Beyond that, FF faults just once at the start of each phase, where a phase is as defined in section 3. Since each phase has length at least k, there are at most $\lceil\frac{\tau_{\max}-k}{k}\rceil$ phases after the first k requests. Hence FF faults at most $k+\lceil\frac{\tau_{\max}-k}{k}\rceil$ times until time $\tau_{\max}+1$.

This gives a lower bound of $\frac{\tau_{\max}}{k+\lceil\frac{\tau_{\max}-k}{k}\rceil}$ on the competitive ratio of A. □

Let $\rho = \lceil\frac{\tau_{\max}}{k}\rceil$. In terms of ρ, the competitive ratio is at least $\frac{\rho k}{\rho+k}$. When ρ is large compared to k, this ratio is close to k.

We derive upper bounds on the competitive ratio of marking algorithms as a function of the maximum expiration time. We defined a new class of marking algorithms below. We analyze their performance using the notion of an *epoch*.

Algorithm 4 MARK2

```
initially cache has k dummy pages, all unmarked;
repeat
   r = next request;
   if r ∈ cache and r is fresh then mark r;
   else begin
        if all pages are marked then unmark all pages;
        evict an unmarked page;
        fetch and mark r;
   end;
```

For an interval $[s,t]$ let P_s^t denote the set of pages requested in that interval.

Definition 2 *An epoch starting at time s ends at time t where t is the maximum time such that $\sum_{p\in P_s^t} \alpha_s^t(p) \leq k$.*

The next epoch begins at time $t+1$. The first epoch begins at time 1.

Theorem 13 *For $\tau_{max} > k$, MARK2 is $\min\{\lceil \frac{\tau_{\max}}{k} \rceil + 1, k\}$-competitive.*

Proof: Let $\rho = \lceil \frac{\tau_{\max}}{k} \rceil$. Divide the input sequence into epochs. Suppose there are N epochs, and assume, for simplicity, that the last epoch was completed. The start of a new epoch coincides with the time when the marking algorithm unmarks all the pages in the cache.

Suppose that epoch i begins at time s and ends at time t. It follows from the definition of epoch that $\sum_{p \in P_s^t} \alpha_s^t(p) = k$. MARK2 constructs a greedy cover for pages in P_s^t, hence it faults at most k times in epoch i. Thus $f_{MARK2}(\sigma) \leq Nk$.

By Lemma 4, any algorithm will need at least $\alpha_s^t(p)$ distinct copies of page $p \in P_s^t$ to serve the requests for p in epoch i. Hence, in total any algorithm will need k distinct copies of pages in P_s^t to service all requests in epoch i. We use this observation to derive a lower bound on $f_{OPT}(\sigma)$.

Let $f_1, \cdots, f_N$ be the number of faults in each epoch, thus $f_{OPT}(\sigma) = \sum_{i=1}^{N} f_i$. Consider the last epoch. At least k distinct copies of pages are needed to serve all the requests in epoch N. Since the algorithm faults f_N times in epoch N, the other $k - f_N$ pages would have been brought in during previous epochs. Note that 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 $N-1, N-2 \cdots N-\rho$, since all pages brought before that would have expired. This implies

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

Similarly we get

$$\begin{aligned} f_{N-\rho-1} + f_{N-\rho-2} + \cdots + f_{N-2\rho-1} &\geq k \\ &\vdots \\ f_\ell + \cdots + f_1 &\geq k \end{aligned}$$

Adding all these equations we get

$$f_{OPT}(\sigma) \;\geq\; \left\lceil \frac{N}{\rho+1} \right\rceil k \;\geq\; \frac{Nk}{\rho+1} \tag{6}$$

Hence the competitive ratio of MARK2 is at most $\frac{Nk}{(Nk)/(\rho+1)} \leq \rho + 1$. By analysis similar to that used for MARK1 in section 3, we can also show that the competitive ratio is also bounded by k. Hence the competitive ratio is bounded by $\min\{\rho+1, k\}$. □

MARK2 seems rather wasteful since it might evict a fresh page even though some other page in the cache is stale. However, we are unable to show an improved competitive ratio for any deterministic algorithm.

6 The Offline Problem

The offline problem of caching with expiration times can be stated as follows:
Input: The cache size k, a request sequence σ of length n, a sequence of expiration times τ that satisfy the monotonicity assumption.

Output: A sequence of pages to evict such that there is a fresh copy of page σ_i in the cache at time i and the number of faults is minimized.

We showed in Theorem 6 that the natural modification of Farthest in the Future is not optimal. We have not found an optimal offline algorithm polynomial in both n and k. We do not know if the problem is NP-complete for arbitrary k.

Theorem 14 *There is an optimal offline algorithm that runs in time $n^{O(k)}$.*

Proof: The problem of finding the optimal offline algorithm can be modeled as a shortest path problem on a graph with $n^{O(k)}$ vertices. 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 corresponding to the subsets of pages currently in the cache. Hence the total number of nodes is bounded by n^{k+1}.

A configuration at time t is connected to those configurations at time $t+1$ which can be reached from it. 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 configurations at time n 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 $n^{O(k)}$. □

Theorem 15 *The is a factor-3 approximation to the offline problem that runs in time poly(n, k).*

Proof: We define an offline algorithm OFF which proceeds in phases like the randomized marking algorithm RMA. We follow the terminology used in Section 4. At the start of phase i, since algorithm OFF knows the entire request sequence, it can identify the sets A_i, B_i and C_i. It evicts all pages in A_i and then the c_i pages which are not requested in this phase. These c_i pages are the only pages that could be fresh when evicted. Note that there are only c_i new pages are requested in phase i. So we do not need to evict any page in B_i.

Suppose that Algorithm OFF evicts x fresh pages and y stale pages.

$$\begin{aligned} y &\leq \sum_p \alpha(p) \leq f_{OPT}(\sigma) && \text{By Lemma 2} \\ x &= \sum_i c_i \leq 2f_{OPT}(\sigma) && \text{By Equation 2} \\ \Rightarrow F_{OFF}(\sigma) &= x+y \leq 3f_{OPT}(\sigma) \end{aligned}$$

Hence OFF is a factor 3 approximation to the offline problem. □

Corollary 16 *Algorithm FF2 gives a factor-3 approximation to the offline problem.*

Proof: We analyze FF2 in phases. Assume that during phase i, algorithm FF2 faults on a request to page p and evicts a fresh copy of page q. Then it must be that q is requested farthest in the future and the $k-1$ other pages currently in the cache are requested before q. Also page p has been requested in phase i. Since phase i terminates just before the $(k+1)^{st}$ distinct page is requested, we conclude that page q is not requested again in phase i.

Thus if FF2 evicts a live page, that page is not requested again in the same phase. Hence the analysis of Theorem 15 can be applied to it implying that FF2 gives a factor-3 approximation to the offline problem. □

7 Further Work

There is a gap between the upper and lower bounds on the competitive ratio in Theorems 12 and 13. 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. Finally, the status of the optimal offline problem (NP-completeness or upper bounds polynomial in both n and k) is open.

References

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

[2] 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

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

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

[5] 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.

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

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

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

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

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

[11] 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.

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

[13] P. Gopalan, H. Karloff, A. Metha, M. Mihail and N. Vishnoi, Caching with EXpiration Times, Proceedings of SODA 2002, pp 540-547.

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

[15] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing Data Cubes Efficiently, Best Paper Award, *Proceedings SIGMOD* 1996, pp 205-216.

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

[17] S. Irani. Competitive Analysis of Paging: A Survey. *Available online from http://www.ics.uci.edu/ irani/Online/Online.html*

[18] 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.

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

[20] Kimbrel, T., "Online Paging and Caching with Expiration Times", Theoretical Computer Science 268 (2001), pp. 119-131.

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

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

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

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

[25] J. Ullman. Efficient Implementation of Data Cubes Via Materialized Views, a survey of the field for the 1996 *KDD Conference*.

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