Exploiting Regularities in Web Traffic Patterns for Cache Replacement

Edith Cohen Haim Kaplan

AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932 USA
{*edith,hkl*}*@research.att.com*

Abstract

Caching web pages at proxies and in web servers' memories can greatly enhance performance. Proxy caching is known to reduce network load and both proxy and server caching can significantly decrease latency. Web caching problems have different properties than traditional operating systems paging, and cache replacement can benefit by recognizing and exploiting these differences. We address two aspects of the predictability of traffic patterns: the overall load experienced by large proxy and web servers, and the distinct access patterns of individual pages.

We formalize the notion of "cache load" under various replacement policies, including LRU and LFU, and demonstrate that the trace of a large proxy server exhibits regular load. Predictable load allows for improved design, analysis, and experimental evaluation of replacement policies. We provide a simple and (near)-optimal replacement policy when each page request has an associated distribution function on the next request time for the page. Without the predictable load assumption, no such online policy is possible and it is known that even obtaining an offline optimum is hard. For experiments, predictable load enables comparing and evaluating cache replacement policies using *partial traces*, containing requests made to only a subset of the pages.

Our results are based on considering a simpler *interval caching* model for cache replacement. We relate traditional and interval-caching policies under predictable load, and derive (near)-optimal replacement policies from their optimal interval-caching counterparts.

1 Introduction

Caching of web pages at different web entities can greatly enhance performance. Proxy caching is known to reduce network load and both proxy and server caching (keeping frequently requested pages in memory) can significantly decrease user-perceived latency. Cached resources and traffic patterns in the web environment exhibit different characteristics than in traditional operating systems paging. One well-recognized difference is the variability in sizes and fetching costs of different resources (pages), and indeed, replacement policies that incorporate these parameters were suggested and evaluated [7, 13, 21, 20].

As is the case in traditional paging, individual requests remain hard to predict. We focus on two separate predictable aspects of the traffic in the web environment. The first is the usage levels experienced by large proxy and (typically) web servers. The second is distinct per-page characteristic distributions on the inter-request times for the page. We elaborate on these two phenomenon and exploit them, separately and together, for improved cache replacement.

Large proxy caches serve large group of users with a large group of independent resources. This diversity results in statistical behavior and therefore regular usage patterns. In their analysis of Berkeley home IP traces, Gribble and Brewer [12] observed burstiness of client activity on shorter time scales (tens of seconds), but predictable activity levels in larger time scales and the presence of a definite diurnal cycle. Arlitt and Williamson [2] observed similar regularity in client activity levels in web servers. Balakrishnan et al. observed regularity in their in depth analysis of traffic to the web server of the Atlanta Summer Olympic Games [3]. These studies suggest predictable patterns in the effectiveness level of the cache. Most cache replacement policies evict a page which maximizes a particular *eviction function*. LRU (least recently used), for example, evicts the page with largest passed time since its last request, LFU (least frequently used) evicts the page with lowest number of requests,[1] and the offline optimal Belady's algorithm [4] evicts the page to be requested furthest in the future. Intuitively, the *cache load* is captured by a value of the eviction function which is below that of evicted pages but above that of most cached pages. We explore the load function under various cache sizes and replacement policies on proxy traces recorded from the AT&T Research web proxy. Our experiments demonstrate that cache load levels are indeed predictable and exhibit a definite diurnal cycle. When the load is steady, we refer to the cache as being in a *steady state*. When the load varies, we capture it by a weight function $\pi(t) \geq 0$ which prices higher times of high demand. The premise of *predictable load* is that an appropriate weight function exhibits diurnal cycles and therefore is predictable.

A second phenomenon we address is patterns of requests to individual pages. Reference locality between accesses to different pages [14, 6, 11] was observed in web traffic [10, 19, 5]. Studies [7, 20] which examined the distribution of lengths of inter-request time intervals between requests for the same page confirm high correlation between time passed since last request and likelihood of another request, which supports the use of LRU (Least Recently Used). The extent of this correlation, however, is highly dependent on the requested URL. Further studies observe that pages tend to have *distinct characteristic distributions of inter-request times*, which supports the observed success of the LFU (Least Frequently

[1] the eviction value may be normalized by elapsed time

Used) replacement policy. We propose a replacement policy that refines both LRU and LFU, and utilizes statistics on inter-request times of each page. Our proposed policy is simple and (nearly) optimal under steady state or predictable load. In contrast, known polynomial time algorithms that do not assume predictable load have considerably weaker performance guarantees.

The underlying problem is modeled as cache replacement with *inter-request distributions (IRD caching)*: each request in a sequence is provided with a distribution function on the time till the next request for the requested page. This model[2] was first considered by Lund et al. [17] in the context of holding time policies for virtual circuits (for routing IP traffic over ATM [15]) and was termed IP-paging. The same model was proposed later on for web caching [18, 9]. Following Lund et al. [17], our "yard-stick" is the optimal online algorithm which makes best use of knowing the underlying probability space, but does not know the actual deterministic sequence apriori. This algorithm is well defined by Markov decision theory for discrete distributions, but is exponential in the number of requests and description of the distributions [17]. As far as we know, it is still not known if it can be obtained in polynomial time or have a polynomial time description. Lund et al. proposed the 5-competitive median algorithm as an implementable alternative.[3] Hence, predictable load allowed for a replacement policy for IRD-caching with much stronger performance guarantee than possible otherwise.

We obtain our results using an alternative caching model that we call *interval caching*. We propose and substantiate interval caching as a general experimental and analytic tool. The input for an interval caching algorithm is a sequence of page requests. For each request, the algorithm decides on a maximal time interval for which the requested page is cached. The page is cached until requested again or until its caching time expires. (Note that there is no fixed limit on the cache size.) Cache replacement policies have analogous interval-caching policies, whose actions are consistent with the eviction function of the replacement policy. Namely, the eviction value of an expired page is always no lower than that of cached pages. Predictable load can be incorporated in the analogous interval-caching policies, and result in decisions that mimic the cache replacement algorithms. For example, An interval-caching policy analogous to LRU would keep pages cached for a fixed amount of time passed their most recent request. When LRU is in steady state, which roughly means that items spend approximately the same amount of time in the cache, the analogous policy applied with the right caching window simulates LRU with high accuracy.

Interval-caching is important for experimental comparison and evaluation of replacement policies when only *partial traces*, which contain requests made to a subset of the pages, are available. Evaluation using a standard model necessitates a full workload for the cache. Interval-caching evaluation is performed using the analogous policies, and considers trade-offs of hit-rate and average cache size. One example of such experiment is the evaluation of server-assisted cache replacement [9]. In server-assisted cache replacement, the web server collects statistics on access patterns to its pages, and disseminates it as hints to its clients that enhance cache replacement in the clients' caches. Full-workload evaluation of server-assisted vs. other replacement policies requires a client (proxy) trace along with server traces for each web server the client used, which is infeasible experimentally. Interval-caching evaluation, in contrast, can be performed using a server trace, which provides for each client only its requests made to the particular server. Interval-caching was also employed by Keshav et al. [16, 15] to evaluate holding time policies for IP over ATM. Here we provide the theoretical foundations for interval-caching evaluation.

The interval-caching model often admits simple optimal policies in situations where no such standard replacement policies are known. We propose a general technique of transforming interval-caching optimal policies into (nearly) optimal replacement policies, under predictable load. Our methodology is to consider the optimal algorithm in the interval-caching model, construct the analogous replacement policy, and prove (near)-optimality of the replacement policy when the cache load is predicted (nearly) correctly.

We first apply this technique in a model where with each request we are provided with the next request time for the requested page. Future requests for new pages, however, are not provided apriori. When page sizes and fetching costs are uniform, this information suffices to do optimally, by simply evicting the page to be requested furthest in the future [4]. When fetching costs vary, however, this information is insufficient [13]. Intuitively, with uniform costs there is a total order on the value of all cached pages (according to their next request time). With varying costs, the relative value of cached pages depends on future requests for pages not yet seen. As a simple example, suppose that the replacement algorithm needs to evict one of two pages: The first page yields $10 per hit and will be requested in 5 minutes, and the second will yield $1 in 2 minutes. The "correct" decision depends on the number and type of future requests received between 2 and 5 minutes from now: the first page should be cached unless we expect it to be evicted before its next request. This example demonstrates that without further assumptions, there is no near-optimal replacement policy. Our proposed algorithm incorporates the load curve, and we prove that it is optimal if the load is predicted correctly. The offline problem of caching with varying sizes and fetching costs can be formulated as an integer-program. When sizes are uniform, the LP relaxation admits an integral solution, and hence, a polynomial-time solution. When page sizes vary, the offline problem becomes NP-complete [13], but an approximate solution can be obtained by rounding the LP-relaxation [8, 1].

This frameworks yields a near-optimal algorithm in the more general (and more realistic) IRD caching model. A simple example demonstrates that cached pages with different distribution function are "incomparable," even when pages have uniform sizes and fetching costs. In the example, the first page has 50% probability to be requested in 2 minutes and 50% probability to be requested in 4 minutes. A second page will be requested next in 3 minutes. The "correct" eviction decision obviously depends on new requests received 2–4 minutes from now. Hence, an optimal online algorithm is not possible without predictable load or further assumptions.

For simplicity of presentation, we first focus on steady state and defer the details of how to extend our results to predictably-varying load to Section 7. Most proofs are omitted due to space limitations.[4]

The rest of the paper is organized as follows. Section 2 defines steady state and the interval caching model. In Section 3 we provide experimental evidence to the steady state and predictable load assumptions for LRU, LFU, and OPT (Belady) replacement policies. We conducted our experiments on a large proxy log. Section 4 substantiates the relation between the interval caching and regular models for LRU and LFU. Sections 5 and 6 use the predictable load assumption, and the interval-caching model, to provide near optimal

[2] with uniform fetching costs and sizes

[3] with respect to the *miss rate*. The median algorithm does not have bounded competitive ratio with respect to the hit rate. Our algorithm is (nearly) optimal with respect to *hit rate*.

[4] A full version can be found at `http://www.research.att.com/~edith/papers/webcache.ps.Z`

replacment policies in settings where no optimal online replacement policy exists otherwise. In Section 5 we consider varying sizes and fetching costs in a model where the next request time is known. In Section 6 we extend it to a model when only a probability distribution on the next-request time is available.

2 Preliminaries

Our input is a sequence of requests for web pages. Each request has an arrival time associated with it. Additional information about future requests may be given with each request (such as the time of the next request for the same page or a probability distribution for the time of the next request for the same page). Each page has size and profit (fetching cost) associated with it. We shall assume that fetching costs and sizes are all one unless we explicitly state otherwise.

A *cache replacement algorithm* gets the requests on-line and for each request for a page that is not in the cache decides whether to put the page in the cache. In case it decides to cache the page and the cache is full, (i.e. the sum of the sizes of the pages currently in the cache and the size of the new page exceeds k) it also has to decide which page (pages) to evict in order to make room for caching the new page. If a page requested is in the cache then the algorithm gains the profit associated with the requested page. The profit of the algorithm on a sequence is the sum of the profits the algorithm gained through requests to cached items. We shall assume that the cache size is an additional parameter to the cache replacement algorithm, so the profit of an algorithm A is a function of the request sequence s and the cache size k, and denoted by $p_A(s,k)$. Let T be the elapsed time of the sequence s. Since we use cache of size k for T seconds we define the cost of A on s to be $c_A(s,T) = k * T$. We shall consider only algorithms A such that for every s, $p_A(s,k)$ is a nondecreasing function of k. It is easy to see that our definition captures all standard cache replacement algorithms.

2.1 Interval Caching

We define an *interval caching algorithm* H as follows. Given a request r for a page u_r (u_r can be in the cache or outside of the cache) the algorithm decides on a maximal time interval $\Delta_H(r)$ for which it caches u_r. The page u_r remains in the cache until either $\Delta_H(r)$ expires, or until it is requested again (and then a new value $\Delta_H(r')$ is generated by the algorithm for the page $u_{r'} \equiv u_r$). Note that the interval caching model does not involve cache capacity constraints.

If a page is requested again before the time interval allocated to it in its previous request expires, then the algorithm gains the profit associated with the page. The *profit* of the algorithm is the sum of the profits gained from individual requests. The *cost* of the algorithm is the sum of the lengths of time intervals during which pages were actually cached. We shall assume that the algorithm takes an additional parameter V so the cost, $c(s,V)$, and profit, $p(s,V)$, are functions of V as well as of the request sequence s. Note that the cost measure for interval-caching algorithms generalizes cache capacity.

2.2 Eviction function

Most standard cache replacement algorithms evict the page that maximizes the value of a particular function. For example LRU uses the function whose value for a page u is the time since u was last requested. Notice that this function is not unique as for example, we can always get an equivalent function by scaling. Sometimes the non-uniqueness is even more subtle: LRU for example is also characterized by the function whose value for a page u is the number of requests since u was last requested. For an algorithm A we denote by $f_A(u,t)$ one particular eviction function that characterizes A's decisions. It would be clear from context which function we refer to. We assume f_A is defined for all pages nomatter whether they are cached at time t or not.

We say that a sequence s is *non-redundant* with respect to a replacement algorithm A if no two different pages have the same eviction value at the same time. (Since the eviction function characterizes the algorithm, the property of being non-redundant is independent of the choice of the eviction function.) On non-redundant sequences the actions of A are uniquely determined.

For LRU, a sequence is non-redundant if no two different requests arrive at the same time. For LFU, a sequence is non-redundant if in every prefix of the sequence no two pages have the same number of occurrences. The latter assumption can be forced by perturbing the "request count" on the first request for each page u (to be $1 \pm \epsilon_u$).

We shall assume throughout the paper that the input sequence is non-redundant with respect to the cache replacement algorithm in question.

2.3 Steady state and predictable load

Let A be a cache replacement algorithm with a corresponding eviction function f_A, and let $u_{\max}(t)$ be the item in the cache that maximizes f_A at time t. The value $f_A(u_{\max}(t),t)$ captures the actions of the replacement policy. Item u remains in cache as long as $f_A(u,t) \le f_A(u_{\max}(t),t)$. In reality, exact future values of the threshold function $f_A(u_{\max}(t),t)$ are hard to predict, but regularity of traffic patterns and in particular a *steady state* assumption suggest that the future threshold is known approximately. We say that A is in ϵ-*steady state* for a request sequence s and cache size k, if there exists a threshold $C(k)$ such that while processing s with cache of size k, the following conditions hold:

1. If A decides not to cache a page u at time t then $f_A(u,t) > C(k)$.
2. If A evicts a page u at time t then $f_A(u,t) > C(k)$.
3. For every t and for every page u' such that $|\{q | f_A(q,t) \le f_A(u',t)\}| \le (1-\epsilon)k$, $f_A(u',t) \le C(k)$. To simplify the presentation we assume that $(1-\epsilon)k$ is an integer here and throughout the paper.

From our applications point of view it would be important that the cache replacement algorithm is in steady state for a range of cache sizes $[K, K']$, rather than for a single one. Furthermore, thresholds for various cache sizes should be such that $C(k)$ is nondecreasing. When an algorithm A can be characterized by more than one eviction function we shall prefer a choice that facilitates steady state.

If page sizes vary, we have to make two changes to the definition above. First, we add a requirement that the sum of the sizes of cached pages at any one time is at least $(1-\epsilon)k$. Second, we change the third condition as follows. Let s_u denote the size of page u. For every t and for every page u' such that

$$\sum_{\{q | f(q,t) < f(u',t)\}} s_q < (1-\epsilon)k,$$

we have $f(u',t) \le C(k)$.

We assume that the cache is fully initialized with a set of "dummy" pages prior to the first request of s such that the steady state assumption already holds when s starts, and the cache is always full throughout s. Alternatively one can assume that s is in fact a suffix of a longer sequence s' during which the steady state assumption holds. When s starts the cache is already full with pages that were requested in s' before s.

2.4 Optimality

We denote by OPT the cache replacement algorithm that generates maximum profit for any given sequence and cache size assuming that the sequence is known in advance. When fetching costs and sizes are all 1 then it is known [4] that the optimal algorithm would place a page in the cache only if its next request time is smaller then the next request time of a page which is currently cached. The evicted page in that case would be the one with the largest next request time.

We say that an interval caching algorithm A is *optimal* if for every sequence s.

1. $p_A(s,V)$ and $c_A(s,V)$ are nondecreasing functions of V.

2. For every threshold V, there is no algorithm A' and threshold V' such that $c_{A'}(s,V') \leq c_A(s,V)$ and $p_{A'}(s,V') > p_A(s,V)$. (no other algorithm achieves better profit for lesser or equal cost).

Each algorithm A generates a set of profit-cost pairs by sweeping the threshold V. The definition above captures point-wise optimality of each pair. We later introduce the stronger notion of *convex-optimality*, which requires that the upper envelope of the pairs dominates trade-offs obtained by any other algorithm.

The following lemma bounds the performance of the optimal (offline) cache replacement algorithm by the performance of an optimal (offline) interval caching algorithm

Lemma 2.1 *Consider a request sequence s whose elapsed time is T. Let OPT_I be an optimal offline interval caching algorithm and OPT the optimal offline cache replacement algorithm. Then, for cache size $k \leq \lfloor c_{\mathrm{OPT}_I}(s,V)/T \rfloor$ we have that $p_{\mathrm{OPT}}(s,k) \leq p_{\mathrm{OPT}_I}(s,V)$.*

An interval caching policy A_I is *analogous* to a replacement policy A if whenever a page u expires at time t and is removed from the cache by A_I, its corresponding eviction value $f_A(u,t)$ is higher than the eviction values of all currently-cached pages.

For every cache replacement algorithm A that we consider we shall give an analogous interval caching algorithm A_I that generates similar profit-cost trade-offs to A on sequences where A is in steady state.

3 Experimental evidence for regularities in proxy cache load

As discussed in the introduction, large proxy servers typically have regular usage pattern. We observed regularity (and predictability) of the eviction function, for several replacement policies. Namely, for LRU (Least Recently Used), LFU (Least Frequently Used), and the optimal policy OPT (Belady's algorithm). Our experiments were performed on a proxy trace recorded from AT&T research web proxy, which handled about 60,000 requests per day.

Consider LRU and the following eviction function. For a page u we define $f_{\mathrm{LRU}}(u,t)$ to be the number of requests received between the time page u was last requested before t and t. Sort the cached pages at time t by f_{LRU}. Let $u_{\max}(t)$ denote the last page in the sorted order $u_{0.8}(t)$ is the page in position $\lfloor 0.8k \rfloor$ and $u_{0.6}(t)$ is the page in position $\lfloor 0.6k \rfloor$. In Figure 1 we plot three functions of time: $f_{\mathrm{LRU}}(u_{\max}(t),t)$, $f_{\mathrm{LRU}}(u_{0.8}(t),t)$, and $f_{\mathrm{LRU}}(u_{0.6}(t),t)$. Figure 1 shows a daily activity cycle. The variations reflects the presence of reference locality, which depends on the number of active users and the type and level of their activity. Hence, reference locality varies between normal working hours, nights, and weekends.

The eviction function which LFU uses is simply the number of requests issued so far for each page. Let $u_{min}(t)$ be the cached page with the smallest request count at time t. If the next request is for a page u' with a larger request count than the request count of $u_{min}(t)$ then u' replaces $u_{min}(t)$ in the cache. Accordingly, we define $f_{\mathrm{LFU}}(u,t)$ to be the number of requests for p divided by the elapsed time.[5] Figure 2 plots the functions $f_{\mathrm{LFU}}(u_{\min}(t),t)$, $f_{\mathrm{LFU}}(u_{0.8}(t),t)$, and $f_{\mathrm{LFU}}(u_{0.6}(t),t)$, where $u_{\min}(t)$, $u_{0.8}(t)$, $u_{0.6}(t)$ are defined in a similar manner as for f_{LRU}. The plot reflect the discreteness of page frequency counts, since the bulk of cached pages have a handful of requests. This effect is smoothened with time, as the request counts for each page grow. We also notice the affect of a weekend to globally attenuate the value of the eviction function for all pages. Other than these two expected phenomena, the behavior of the cache is very regular.

When each request arrives with the time of the next request for the same page then we can achieve the optimal profit for every cache size. A result of Belady [4] shows that we maximize the profit if we always evict the page whose next request is the furthest in the future. In other words, the value of the eviction function of the optimal algorithm, OPT, on a page is the time until its next request. Figure 3 demonstrates a steady behavior of f_{OPT} as a function of time. The figure shows $f_{\mathrm{OPT}}(u_{0.99}(t),t)$[6], $f_{\mathrm{OPT}}(u_{0.8}(t),t)$, $f_{\mathrm{OPT}}(u_{0.6}(t),t)$ where $u_{0.99}(t)$, $u_{0.8}(t)$, $u_{0.6}(t)$ are as defined above. As seen in the plot, the functions $f_{\mathrm{OPT}}(u_{0.8}(t),t)$ and $f_{\mathrm{OPT}}(u_{0.6}(t),t)$ are very similar for different work days, establishing a predictable behavior.

4 Interval caching counterparts for LRU and LFU

LRU and LFU have the following property: the pages in the cache at any one time are the k smallest[7] with respect to the value of the eviction function. For any algorithm A with this property we define an analogous interval caching algorithm A_I that exhibits similar profit-cost tradeoffs on sequences where A is in steady state.

Definition 4.1 *Let A be a cache replacement algorithm with eviction function f_A such that*

- *The cache always contains the k pages with smallest f_A values.*

- *The eviction value of u at time t, $f_A(u,t)$, may depend on all requests seen so far,* up till the most recent request for *u, and on the current time t.*

The corresponding analogous interval caching algorithm, A_I, is as follows

- *After serving a sequence s at time t and upon a request r for page u_r, A_I chooses $\Delta(r)$ to be the maximum such that $f_A(u_r,t') \leq V$ for every $t' \leq t+\Delta(r)$.*

[5] Note that the division by elapsed time is for normalization of the eviction function and does not change the replacement policy.

[6] We use $f_{\mathrm{OPT}}(u_{0.99}(t),t)$ here instead of $f_{\mathrm{OPT}}(u_{max}(t),t)$ because the curve is smoother: the curve $f_{\mathrm{OPT}}(u_{max}(t),t)$ has a very large number of very long inter-request intervals.

[7] technically for LFU, these are the k largest pages, but then we consider the negated eviction function.

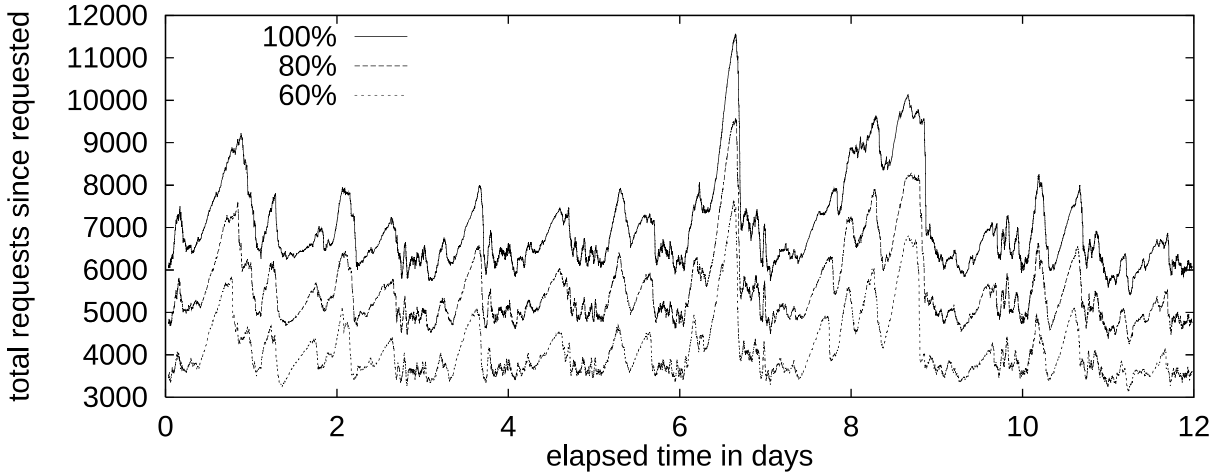


Figure 1: The (minimum) number of requests that have arrived since a cached page was last requested, for LRU with cache size 5000. The line marked 100% is $f'_{\mathrm{LRU}}(u_{\max}(t), t)$. The other two lines are $f'_{\mathrm{LRU}}(u_{0.8}(t), t)$, and $f'_{\mathrm{LRU}}(u_{0.6}(t), t)$.

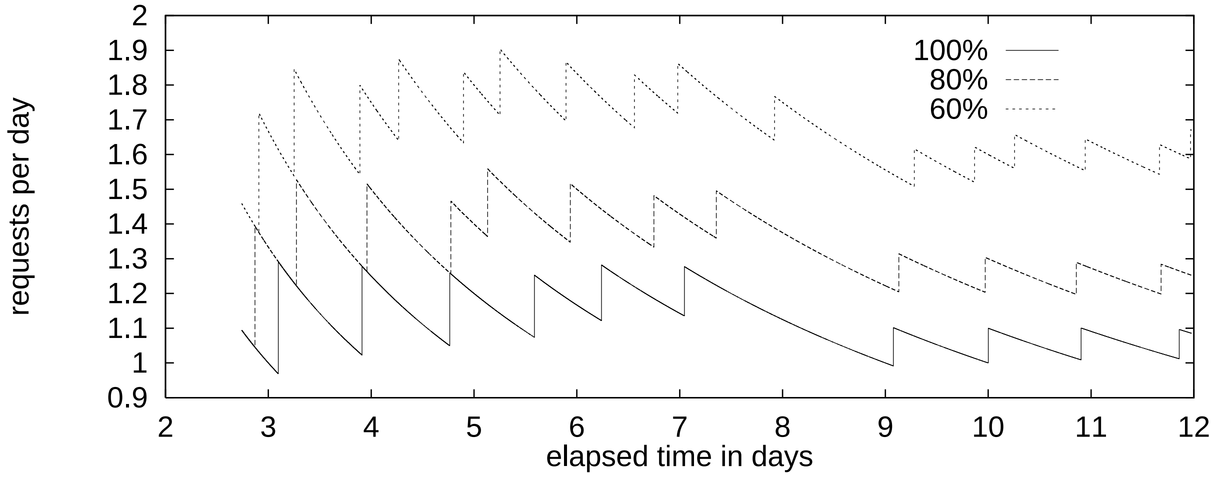


Figure 2: Request rate as a function of time for items in the cache during LFU. The line marked 100% shows is $f_{\mathrm{LFU}}(u_{\min}(t), t)$, the line marked 80% shows $f_{\mathrm{LFU}}(u_{0.8}(t), t)$, and the line marked 60% shows $f_{\mathrm{LFU}}(u_{0.6}(t), t)$.

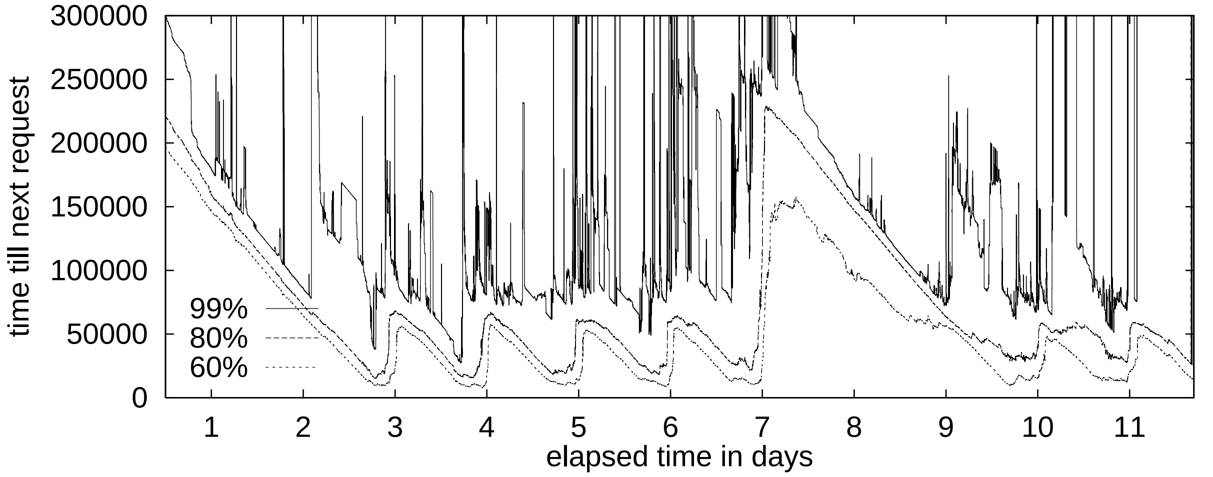


Figure 3: Time to the next request for items in the cache during OPT. The line marked 99% shows is $f_{\mathrm{OPT}}(u_{0.99}(t), t)$, the line marked 80% shows $f_{\mathrm{OPT}}(u_{0.8}(t), t)$, and the line marked 60% shows $f_{\mathrm{OPT}}(u_{0.6}(t), t)$, all with respect to cache size 5000.

Intuitively, the second condition on the eviction function of A means that the eviction value of a page is not affected by requests made to other pages, past its most recent request. This condition enforces "predictability" of the eviction value of a page u, between requests for u.

Trade-offs of profit and cost for A can be obtained by sweeping the cache size k. Similarly, trade-offs for A_I are obtained by sweeping the parameter V. The following theorem establishes that when A is in steady state, the algorithms A and A_I exhibit similar trade-offs.

Theorem 4.1 *Let A and A_I be a cache replacement algorithm and its analogous interval-caching algorithm as above. For all sequences s such that A is in ϵ-steady state for cache size k.*

$$\begin{array}{ccccc} (1-\epsilon)k*T & \leq & c_{A_I}(s, C(k)) & \leq & k*T \\ p_A(s,(1-\epsilon)k) & \leq & p_{A_I}(s, C(k)) & \leq & p_A(s,k) \end{array},$$

where T is the elapsed time of the request sequence.

The interval caching algorithm LRU_I that corresponds to LRU is the algorithm that caches every request for an interval of length V. From Theorem 4.1 we obtain that when LRU is in steady state, its profit to cost trade-offs are similar to those of LRU_I.

We use a normalized eviction function for LFU: We define $f_{\text{LFU}}(u,t)$ to be the number of request for u issued so far, divided by t. The corresponding interval caching algorithm, LFU_I caches u for an interval of length Δ, where Δ is maximum such that the number of occurrences of u in s divided by $t+\Delta$ is still no less than V. Theorem 4.1 implies that when LFU is in steady state, LFU and LFU_I have similar performance.

5 Caching with varying file sizes and fetching costs

We consider paging where pages have varying sizes and costs. We assume that along with page requests we are provided with the next request time of the currently-requested page.

We first state the interval-caching policy GOPT_I and establish its optimality. We proceed and state the analogous replacement policy GOPT. Last, we establish that GOPT is (nearly) optimal, when it is in ϵ-steady state.

Let s be a sequence of page requests such that for $r \in s$, page u_r of size $0 < s_r < \infty$ and fetching cost $0 < f_r < \infty$ is requested at time t_r $(r = 1, 2, \ldots)$. We assume (through Section 5) that each request $r \in S$ arrives with the time Δ_r until the next request for u_r.

In these settings an interval caching algorithm B upon receiving a request r has to decide whether to cache u_r for Δ_r time or not to cache it at all (note that we can assume that these are the only valid choices). Let r be a request for a page u_r. The profit of caching u_r for time length Δ_r following r is $\text{profit}(r) = f_r$, and the corresponding cost is $\text{cost}(r) = s_r\Delta_r$. For a set of requests A we define $\text{profit}(A) = \sum_{r\in A}\text{profit}(r)$ and $\text{cost}(A) = \sum_{r\in A}\text{cost}(r)$. Thus if $S_B \subset S$ is the set of requests where B decided to cache the corresponding page. The profit of the algorithm B on the sequence S is $\text{profit}(S_B)$ and its cost is $\text{cost}(S_B)$.

5.1 The algorithm GOPT_I

Define the utility of request r to be $\text{util}(r) = \frac{\text{profit}(r)}{\text{cost}(r)}$. The interval caching algorithm GOPT_I with a threshold V caches u_r for Δ_r if and only if $1/\text{util}(r) \leq V$.

Note that cost and profit of GOPT_I are nondecreasing step functions of V. The set of (cost, profit) pairs obtained by sweeping V is discrete and its size is at most the number of requests. Furthermore, all points lie on its upper envelope, which is a concave monotonically increasing function.

5.2 Optimality of GOPT_I in the interval-caching model

Consider the set of all profit-cost pairs obtained by GOPT_I with all possible threshold values. It is easy to see that not every possible profit value is achieved by a point in this set. To compensate for that we consider a stronger notion of optimality which we call *convex optimality*. We show that GOPT_I generates sufficiently many (cost, profit) pairs such that it is convex optimal.

An interval caching algorithm A is *convex optimal* if for every sequence s.

1. $p_A(s,V)$ and $c_A(s,V)$ are monotonically nondecreasing functions of V.
2. For every three points $(c_A(s,V_1), p_A(s,V_1))$, $(c_A(s,V_2), p_A(s,V_2))$, $(c_A(s,V_3), p_A(s,V_3))$, where $V_1 \leq V_2 \leq V_3$, and $0 \leq \alpha \leq 1$ such that $c_A(s,V_2) = (1-\alpha)c_A(s,V_1) + \alpha c_A(s,V_3)$, the profits are such that $p_A(s,V_2) \geq (1-\alpha)p_A(s,V_1) + \alpha p_A(s,V_3)$.
3. For every algorithm A' and threshold V' there exists two thresholds V_1 and V_2 and $0 \leq \alpha \leq 1$ such that $c_{A'}(s,V') = (1-\alpha)c_A(s,V_1) + \alpha c_A(s,V_2)$ and $p_{A'}(s,V') \leq (1-\alpha)p_A(s,V_1) + \alpha p_A(s,V_2)$.

The following lemma follows immediately from our definitions.

Lemma 5.1 *If an interval caching algorithm is convex optimal then it is also optimal.* ■

The following theorem establishes that GOPT_I yields optimal trade-offs of profit and cost.

Theorem 5.1 *Algorithm GOPT_I is convex optimal interval caching algorithm.*

Combining Theorem 5.1 and Lemma 5.1 we obtain the following corollary.

Corollary 5.1 *Algorithm GOPT_I is optimal interval caching algorithm.* ■

5.3 The analogous replacement algorithm GOPT

The following replacement algorithm GOPT is analogous to GOPT_I. If a cache miss occurs when requesting a page u, algorithm GOPT compares $1/\text{util}(u)$ to maximum inverse-utility of a page in the cache. If $1/\text{util}(u)$ is smaller, then GOPT evicts the page with maximum inverse-utility value (i.e. with minimum profit to cost ratio) and caches u instead.

5.4 (ϵ-)Optimality of GOPT when in steady state

Theorem 5.2 *Assume that GOPT is in ϵ-steady state for a sequence s and cache size k. Then,*

$$\begin{array}{ccccc} (1-\epsilon)k*T & \leq & c_{\text{GOPT}_I}(s,C(k)) & \leq & k*T \\ (1-\epsilon)p_{\text{OPT}}(s,k) & \leq & p_{\text{GOPT}_I}(s,C(k)) & \leq & p_{\text{GOPT}}(s,k) \\ p_{\text{OPT}}(s,(1-\epsilon)k) & \leq & p_{\text{GOPT}_I}(s,C(k)) & & \end{array}$$

where OPT is the optimal replacement algorithm and T is the elapsed time of the request sequence.

The following corollary establishes that GOPT is nearly optimal when in steady state.

Corollary 5.2 *Under the conditions of Theorem 5.2 we have* $(1-\epsilon)p_{\text{OPT}}(s,k) \leq p_{\text{GOPT}}(s,k)$.

6 Caching with inter-request distributions (IRD caching)

In reality, the time of the next request is not available. In many situations, however, it is possible to obtain an estimate of the next request time in the form of a probability distribution [17]. In the web context, for example, such a distribution can be approximated by the histogram of inter-request times for the page (as seen by the web server or locally by the client's cache) [9]. To model this scenario we relax the assumption that future access times are given exactly and assume that each request arrives together with a probability distribution for the length of the time interval until the next request for the same page.

Specifically, our input is a sequence of page requests $r = 1, \ldots,$ such that page u_r is requested at time t_r. Each request r arrives with a distribution function F_r, and the subsequent request time for u_r is drawn according to this distribution. We shall assume that all request sequences have the same elapsed time T. Also, we allow nonzero probability to the event that a page is never requested again (before T). The sequence of requests includes both new pages and subsequent requests to previously-requested pages. The distribution F_r and the arrival times of "new" pages depends on the "history" of past requests and their associated parameters, but not on the actions taken by the replacement algorithm. These assumptions specify properties of an underlying probability space from which the request sequences are drawn.

We measure a cache replacement algorithm A working with cache size k by its expected profit, $E(p_A(k))$, over the underlying probability distribution of request sequences. The optimal algorithm in this settings, which we shall also denote by OPT is the algorithm which maximizes the expected profit among all algorithms that know the underlying probability space over request sequences, but do not know actual future request times. In this sense, LRU is optimal when the probability density functions are identical for all requests and are non-increasing with time, and LFU (Least Frequently Used) is optimal when each page is requested with a fixed frequency (assuming uniform fetching costs and sizes). Experiments show that per-page inter-request times distributions vary considerably between web sites and for different pages within a site (see for example Figure 4). The variability suggests that better results can be obtained by incorporating the inter-request times distributions in the replacement policy.

Lund et al. observed that the optimal cache replacement algorithm for IRD caching can be determined by solving a linear program of size exponential in the description of the probability space [17]. It is not known whether the optimal strategy has a description of polynomial size.

Here we provide a cache replacement algorithm whose performance is close to optimal under steady state conditions[8]. We start by describing an algorithm MPG_I that generates optimal tradeoffs in the interval caching model. Then we give an analogous cache replacement algorithm, MPG (*Minimum Profit Gradient*), and establish that MPG is nearly optimal, when in steady state.

Earlier we discussed steady state with respect to a single request sequence. We extend the definition and require that "most" sequences (from the underlying probability space) are in steady state and that these sequences are responsible for "most" of the profit.

For simplicity of presentation, the rest of this section assumes uniform fetching costs and sizes. The results can be extended to the general case in a straightforward manner.

[8] The results generalize to predictable load.

6.1 The analogous interval caching model

In the current settings we measure an interval caching algorithm B with a threshold V by its *expected* profit, $E(p_B(V))$, and its *expected* cost, $E(c_B(V))$, over the probability distribution of request sequences.

The probability distribution over request sequences defines a probability distribution over triples constituting of request time, requested page, and associated distribution function. Formally, let S be a set of "possible requests" (triples) each request $r \in S$ has probability q_r that a page u_r is requested in time t_r with associated distribution function F_r. The probability of each triple q_r is the sum over sequences in which such request is made, of the probability of the sequence. It is possible to summarize the actions of B by a mapping from all possible requests S (triples (u_r, t_r, F_r)) to a probability density function $\Delta_{r,V}(t)$ that captures the likelihood that upon encountering a request at time t_r with associated probability F_r, B caches u_r for time period t.

Let r be a request for a page u_r with associated distribution function $F_r(x)$. The expected profit of caching u_r for time length t following r is $\text{Eprofit}_r(t) \equiv F_r(t)$ and the corresponding expected cost is $\text{Ecost}_r(t) \equiv \int_0^t (1 - F(x))dx$. (We shall omit the subscript r when it would be clear from context.) It follows from linearity of expectation that the expected profit of B is

$$E(p_B(V)) = \sum_{r \in S} q_r \int_0^\infty \Delta_{r,V}(t)\text{Eprofit}_r(t)dt$$

and the expected cost is

$$E(c_B(V)) = \sum_{r \in S} q_r \int_0^\infty \Delta_{r,V}(t)\text{Ecost}_r(t)dt \ .$$

6.2 The interval-caching algorithm MPG_I

We denote by

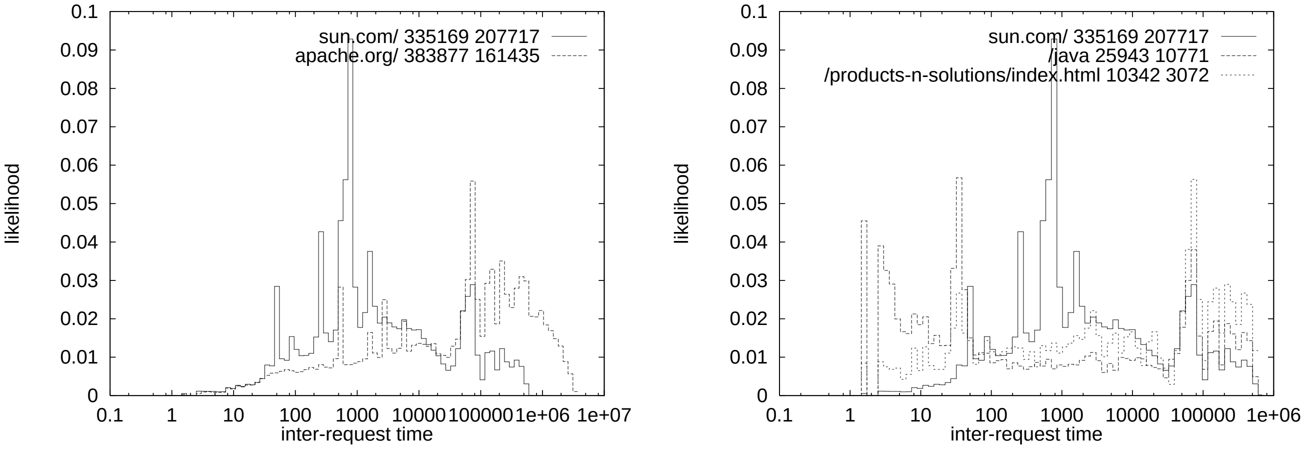
$$\text{util}_r(t_1, t_2) \equiv \frac{\text{Eprofit}_r(t_2) - \text{Eprofit}_r(t_1)}{\text{Ecost}_r(t_2) - \text{Ecost}_r(t_1)} \equiv \frac{F_r(t_2) - F_r(t_1)}{\int_{t_1}^{t_2} (1 - F_r(x))dx}$$

the expected profit gain of increasing the caching period from t_1 to t_2 divided by the expected cost of this increase.

Algorithm MPG_I caches u_r for an interval of length Δ_r where $\Delta_r = \max\{z | \forall y < z \ \ \text{util}_r^{-1}(y, z) \leq V\}$. That is, Δ_r is the maximum point such that for every $y < \Delta_r$, the expected cost of increasing the caching period from y to Δ_r divided by the expected profit gain from this increase is no greater than V.

Remark: Keshav et al. [15] also considered interval caching algorithms. In our terminology, for a fixed cost $\mathcal{O}$ of a cache miss, they define the *overall cost* of a policy A as the total time items are cached plus the total cost of cache misses. It is straightforward to check that given a probability space on request sequences algorithm $\text{MPG}_I(\mathcal{O})$ minimizes the overall cost among all algorithm that know the probability space. Hence MPG_I is identical to the *adaptive policy* of [15]. MPG_I was also used in [9] to conduct experiments in a setup where the request sequence is not completely known. ■

Given a distribution function $F(t)$, there is a discrete set of time intervals that corresponds to choices that MPG_I can make (on the range of $V \geq 0$ threshold values) for caching a page u_r with associated distribution $F_r(t) \equiv F(t)$. We partition the domain of $F(t)$ into intervals $I_1 = [0, t_1]$, $I_2 = (t_1, t_2]$, $I_3 = (t_2, t_3], \ldots$ where t_1 is the maximum $z > 0$ that



(a) Apache and Sun homepages

(b) Sun - various pages

Figure 4: Inter-request histogram for pages (URL's) using Sun and Apache web-server logs

maximizes the quantity

$$\text{util}(0, z) = \frac{F(z)}{\int_0^z (1 - F(x))dx} \ ,$$

and t_i $(i > 1)$ is the maximum $z > t_{i-1}$ that maximizes the quantity

$$\text{util}(t_{i-1}, z) = \frac{F(z) - F(t_{i-1})}{\int_{t_{i-1}}^z (1 - F(x))dx} \ .$$

We define $\text{util}(I_i) \equiv \text{util}(t_{i-1}, t_i)$. It follows from this definition that $\text{util}(I_i) < \text{util}(I_{i-1})$ for every i. The following lemma shows that MPG_I always caches u_r for one of the time periods $0 \equiv t_0, t_1, \ldots$.

Lemma 6.1 *An equivalent alternative definition for the choice of algorithm* MPG_I *is to cache* u_r *for an interval* $[0, t_i]$ *for the maximum* i *such that* $\text{util}(I_i) \geq \frac{1}{V}$.

6.3 Optimality of MPG_I

The definitions of optimality and convex-optimality extend in a straightforward way to our current probabilistic setup. We modify them to use expectations of profit and cost over the probability space defined on the sequences rather than profit and cost of a particular sequence. Lemma 5.1 clearly holds for our new settings here and the following lemma is analogous to Lemma 2.1.

Lemma 6.2 *Let* OPT_I *be an optimal interval caching algorithm, and let* T *be the elapsed time of the request sequences. Then, for cache size* $k \leq \lfloor E(c_{\text{OPT}_I}(V))/T \rfloor$ *we have that* $E(p_{\text{OPT}}(k)) \leq E(p_{\text{OPT}_I}(V))$, *where* OPT *is the optimal cache replacement algorithm in the regular model.*

Fix a request r with a distribution function F_r. The following property of the intervals I_i is needed to establish the optimality of MPG_I.

Lemma 6.3 *Consider* i *and* $t_{i-1} < t < t_i$. *Let* α *be such that* $\text{Ecost}(t) = (1-\alpha)\text{Ecost}(t_{i-1}) + \alpha\text{Ecost}(t_i)$. *Then,* $\text{Eprofit}(t) \leq (1-\alpha)\text{Eprofit}(t_{i-1}) + \alpha\text{Eprofit}(t_i)$.

The following theorem establishes optimality of MPG_I.

Theorem 6.1 *Algorithm* MPG_I *is convex optimal.*

It is clear from the definition of MPG_I that it satisfies the first two requirements of convex optimality. To prove that the third requirement holds we use the following notion of *domination.* A set of profit-cost tradeoffs is a set of points (c, p) in R^2. We say that a set of tradeoffs X *dominates* another set of tradeoffs Y if for all $(y_1, y_2) \in Y$ there exists $(x_1, x_2) \in X$, $(x'_1, x'_2) \in X$, and $0 \leq \alpha \leq 1$ such that $y_1 = (1-\alpha)x_1 + \alpha x'_1$ and $y_2 \leq (1-\alpha)x_2 + \alpha x'_2$. (This is equivalent to saying that the upper envelope of X is "above" Y.) Note that "domination" is transitive. That is, if X dominates Y and Y dominates Z then X dominates Z. Also note that a set dominates itself and can also be dominated by one of its subsets.

Algorithm MPG_I uses the same caching time interval per distribution function F. That is, the function $\Delta_{r,V}(t)$ assigns positive probability to a single value and this value is the same across all requests r with $F_r \equiv F$. Furthermore, the time period is selected from a discrete set specified above. We first establish that it suffices to assume that B has this property.

By iteratively applying Lemma 6.3 (using transitivity of domination), we obtain that the set of profit-cost tradeoffs of algorithms that use caching values other than $t_0^F, t_1^F, \ldots$ is dominated by the set of profit-cost tradeoffs of algorithms that cache only full intervals.

By iteratively applying Lemma 6.4, (using transitivity of domination), we obtain that the set of profit-cost tradeoffs obtained by algorithms that make several different choices of t_i^F upon encountering a request with distribution F are dominated by the set of profit-cost tradeoffs obtained by algorithms that make a consistent decision upon encountering F.

Lemma 6.4 *Consider an algorithm* B *that for some thresholds* V *upon encountering a request with distribution* F, *selects at least two different caching periods,* t_i^F *and* t_j^F *with* $i < j$ *with positive probabilities. Then* B *is dominated by algorithms that make identical choices to* B *elsewhere, but use at most one of the choices* t_i^F *or* t_j^F *upon encountering* F.

Proof.[of theorem 6.1] To complete the proof we have to show that every algorithm that makes a consistent choice of caching time upon encountering every request with distribution F, and the choice is one of $t_1^F, t_2^F, \ldots$, is dominated by MPG_I. For each distribution F, let $q_F = \sum_{\{r|F_r=F\}} q_r$ be the total weight of requests with associated distribution F. The problem reduces to the following. For each distribution F we have a set of items where the ith item corresponds to caching an item for time interval $I_i^F = (t_i^F, t_{i+1}^F]$. This item has cost $q_F(\text{Ecost}(t_i^F) - \text{Ecost}(t_{i-1}^F))$ and profit $q_F(\text{Eprofit}(t_i^F) - \text{Eprofit}(t_{i-1}^F))$ and its utility is $\text{util}(I_i^F)$. The problem of choosing an optimal set of caching decisions reduces to that of the previous section, under the additional constraint that if we choose I_i^F we have to choose I_j^F for $j < i$. As we had seen, the trade-offs for the convex optimal algorithm GOPT_I are obtained by selecting the subset of items with inverse-utility no greater than a threshold V (and varying the threshold). Since for all F and $j < i$, $\text{util}(I_i^F) < \text{util}(I_j^F)$, all these choices fulfill the additional constraint. Note that these choices correspond to those of MPG_I. ∎

6.4 The replacement algorithm MPG

The cache replacement algorithm that is analogous to MPG_I is the following *minimum profit gradient algorithm* (MPG). For a page u_r, requested at time t_r, with distribution F_r, let $I(u_r, t)$ be the interval of F_r that contains $t - t_r$. The eviction function of MPG is $f_{\text{MPG}}(u_r, t) = \text{util}^{-1}(I(u_r, t))$. When a request for page u at time t generates a cache miss, algorithm MPG compares $f_{\text{MPG}}(u, t)$ with $f_{\text{MPG}}(u_{\max}(t), t)$ where $u_{max}(t)$ is the page in the cache which maximizes $f_{\text{MPG}}(u, t)$. If $f_{\text{MPG}}(u, t) < f_{\text{MPG}}(u_{\max}(t), t)$ then page u replaces page $u_{\max}$ in the cache. As before we assume nonredundancy, i.e. no two cached pages have the same eviction value at the same time.

Prior to establishing that MPG is (nearly) optimal when in steady state, we need to extend the definition of steady state. We defined steady state with respect to a single sequence of requests. In the current context, we have to consider a probability distribution over request sequences, and it may be too strong a requirement to insist that steady state happens with probability 1. Instead, we make a more general and plausible assumption that with probability $1-\delta$ the sequence is in steady state with (the same) parameter $C(k)$. Moreover, we require that these "steady state" sequences contribute to at least $1 - \delta$ of the expected profit of both MPG_I and MPG, and at least $1 - \delta$ to the expected cost of MPG_I. We then say that MPG is in ϵ, δ-steady state. The following theorem show that MPG, OPT, and MPG_I have similar performance when ϵ and δ are small.

Theorem 6.2 *Suppose the replacement policy* MPG *is in ϵ, δ-steady state with cache size k. Then,*

$$(1-\delta)(1-\epsilon)k * T \le E(c_{\text{MPG}_I}(C(k))) \le \frac{k * T}{1-\delta} ,$$

and

$$E\left(p_{\text{OPT}}\left((1-\delta)(1-\epsilon)k\right)\right) \le E(p_{\text{MPG}_I}(C(k))) \le \frac{E(p_{\text{MPG}}(k))}{1-\delta} .$$

The following corollary establishes that MPG is (nearly) optimal when in ϵ, δ-steady state, and ϵ and δ are small.

Corollary 6.1 *Under the assumption of Theorem 6.2*

$$(1-\delta)E(p_{\text{OPT}}((1-\delta)(1-\epsilon)k)) \le E(p_{\text{MPG}}(k)) .$$

7 Predictable non-steady load

Our results outlined so far apply to steady state. Here we extend them to *predictable load.*

We capture the *load* on the cache by a weight function on time $\pi(t) \ge 0$, which we refer to as *price function.* Intuitively, the function π assigns different values to different times, according to the current "value" of cache storage. Times of high usage (when evicted pages have low eviction values), have higher price. Our results generalize in a straightforward manner when the time is "weighted" by a price function (time t is mapped to time $\int_0^t \pi(t)dt$).

For LRU and LFU, prices do not change the replacement policies, but do affect eviction values and the steady state property (sequence being in steady state is prices-dependent). Therefore, the interval caching algorithms LRU_I^π and LFU_I^π that correspond to steady state depend on π. The replacement policies GOPT^π and MPG^π, are price-dependent. Our results extend to show that GOPT^π and MPG^π are nearly optimal when they are in ϵ-steady state with respect to π.

The motivation for the *predictable load* assumption is that even though individual requests cannot be predicted, we can predict a price function such that the execution is in ϵ-steady state with respect to these prices. The predictability premise follows the observed regularity of usage patterns and eviction function values. In the full version we show that for considered replacement policies, for "most" sequences (we exclude redundancies), there exist prices such that the sequence is in steady state. Hence, appropriate prices do exist.

7.1 Incorporating prices

Our derivations can be generalized to incorporate any weight function $\pi(t) \ge 0$. The cost of unit storage during the time interval $[t_1, t_2]$ is $\int_{t_1}^{t_2} \pi(t)dt$ under prices π rather than $|t_2 - t_1|$. We outline some implications of this substitution.

- The cost of a cache replacement algorithm with cache size k is $k\int_0^T \pi(t)dt$ (the start time is $t = 0$).
- The cost incured by an interval caching algorithm in caching a page of size s_r from time t_r until its next request time $t_r + \Delta_r$ is $s_r \int_{t_r}^{t_r+\Delta_r} \pi(t)dt$. Note that the actions of the optimal interval caching policy are price-dependent.

The weighted version of Lemma 2.1 bounds the performance of the optimal replacement algorithm by that of the optimal interval-caching algorithm.

Lemma 7.1 *Consider prices $\pi(t) \ge 0$, and a request sequence s. Let $\text{OPT}_I^\pi(s, V)$ be an optimal interval-caching policy with respect to prices π. Let $\text{OPT}(s, k)$ be the optimal replacement algorithm for cache size k. Then, for cache size*

$$k \le \left\lfloor \frac{c_{\text{OPT}_I^\pi(s,V)}}{\int_{t_0}^T \pi(t)dt} \right\rfloor ,$$

we have

$$p_{\text{OPT}(s,k)} \le p_{\text{OPT}_I^\pi(s,V)} .$$

Note that the lemma holds for any price function π.

7.2 LRU and LFU

Prices can be incorporated into eviction functions f^{π}_{LRU} and f^{π}_{LFU} of LRU and LFU. For a page u previously requested in time t_r we define $f^{\pi}_{\text{LRU}}(u,t) = \int_{t_r}^{t} \pi(t)dt$. If the page was requested c_u times until time t we define $f^{\pi}_{\text{LFU}}(u,t) = \int_0^t \pi(t)dt/c_u$. The relative order of eviction values of pages remains the same for all π, and hence, LRU and LFU are not affected by weighting of time. The steady state property, however, is dependent on the price function, as well as the actions of LRU^{π}_I and LFU^{π}_I. The following lemma shows that for a given sequence there exists a price function for which LRU (LFU) is in ϵ-steady state for a very small ϵ.

Lemma 7.2 *Let s be a non-redundant sequence for* LRU *(*LFU*). There exists a price function $\pi_s(t) \geq 0$ such that* LRU *(resp.,* LFU*) is in ϵ-steady state (with threshold 1) on s, where $\epsilon = 1/k$, if the cache size is k.*

7.3 GOPT^{π} and MPG^{π}

The interval-caching algorithm GOPT^{π}_I is defined as follows. The cost of caching u_r from time t_r to its next request at time $t_r + \Delta_r$ is

$$\text{cost}(r) = s_r \int_{t_r}^{t_r+\Delta_r} \pi(t)dt \ .$$

The utility is defines as before,

$$\text{util}(r) = \frac{\text{profit}(r)}{\text{cost}(r)} \ .$$

The algorithm GOPT^{π}_I with threshold V caches u_r for time Δ_r if and only if $\text{util}^{-1}(r) \leq V$. Otherwise, u_r is not kept in the cache.

In the IRD caching model, the expected caching cost of a page u_r for time z depends on the distribution function F_r and on the price function

$$\text{Ecost}(F_r, z) = \int_0^z \pi(x)(1 - F(x))dx \ .$$

The interval-caching algorithm MPG^{π}_I considers the expected cost as above, but otherwise, the definition of the utility function and the partition into intervals are as described earlier.

The earlier proofs of optimality of GOPT_I and MPG_I extend in a straightforward way to GOPT^{π}_I and MPG^{π}_I. Note that the analogous replacement policies GOPT^{π} and MPG^{π} depend on the price function π. The theorems relating the performance of the interval-caching policies and the analogous replacement policies, and establishing that GOPT^{π} and MPG^{π} are nearly optimal when in ϵ-steady state, also extend in a straightforward way to GOPT^{π}_I, GOPT^{π}, MPG^{π}_I, and MPG^{π}.

The results can be paraphrased as follows: If the prices $\pi(t) \geq 0$ are such that GOPT^{π} is in ϵ-steady state (resp., MPG^{π} is in δ, ϵ-steady state), then GOPT^{π} (resp., MPG^{π}) are nearly optimal.

References

[1] S. Albers, S. Arora, and S. Khanna. Page replacement for general caching problems. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 31–40. ACM-SIAM, 1999.

[2] Martin F. Arlitt and Carey L. Williamson. Internet Web servers: Workload characterization and implications. *IEEE/ACM Transactions on Networking*, 5(5):631–644, October 1997.
http://www.cs.usask.ca/homepages/faculty/carey/research.html.

[3] H. Balakrishnan, S. Seshan, Stemm M., and Katz R. H. Analyzing stability in wide-area network performance. In *Proceedings of the ACM SIGMETRICS '97 Conference*, 1997.
http://HTTP.CS.Berkeley.edu/~stemm/pubs.html.

[4] L. A. Belady. A study of replacement algorithms for virtual storage computers. *IBM systems journal*, 5:78–101, 1966.

[5] Azer Bestavros. Using speculation to reduce server load and service time on the WWW. In *Proceedings of the ACM 4th international conference on information and knowledge management*, 1995.

[6] A. Borodin, S. Irani, P. Raghavan, and B. Schieber. Competitive paging with locality of reference. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*. ACM, 1991.

[7] Pei Cao and Sandy Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California, December 1997.
http://www.usenix.org/events/usits97.

[8] E. Cohen and H. Kaplan. LP-based analysis of greedy-dual-size. In *Proc. 10th ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, 1999. full version in submission.

[9] E. Cohen, B. Krishnamurthy, and J. Rexford. Evaluating server-assisted cache replacement in the Web. In *Proceedings of the 6th European Symposium on Algorithms*, pages 307–319. Springer-Verlag, Lecture Notes in Computer Science Vol. 1461, August 1998.

[10] E. Cohen, B. Krishnamurthy, and J. Rexford. Improving end-to-end performance of the Web using server volumes and proxy filters. In *Proceedings of the ACM SIGCOMM'98 Conference*, September 1998.

[11] A. Fiat and Z. Rosen. Experimental studies of access graph based heuristics: Beating the LRU standard? In *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, 1997.

[12] Steven D. Gribble and Eric A. Brewer. System design issues for Internet middleware services: Deductions from a large client trace. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, Monterey, California, December 1997.
http://www.usenix.org/events/usits97.

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

[14] A. Karlin, S. Phillips, and P. Raghavan. Markov paging. In *Proc. 33rd IEEE Annual Symposium on Foundations of Computer Science*. IEEE, 1992.

[15] S. Keshav, C. Lund, S. Phillips, N. Reingold, and H. Saran. An empirical evaluation of virtual circuit holding time policies in IP-over-ATM networks. *J. on selected areas in communication*, 13, 1995.

[16] S. Keshav and H. Saran. An empirical evaluation of virtual circuit holding times policies in IP-over-ATM networks. In *Proceedings of the IEEE INFOCOM'94 Conference*, 1994.
http://www.cs.cornell.edu/skeshav/papers/hold.ps.

[17] C. Lund, N. Reingold, and S. Phillips. IP over connection oriented networks and distributional paging. In *Proc. 35th IEEE Annual Symposium on Foundations of Computer Science*. IEEE, 1994.
http://www.cs.cornell.edu/skeshav/doc/94/2-16.ps.

[18] Jeffrey C. Mogul. Hinted caching in the web. In *Proceedings of the 1996 SIGOPS European Workshop*, 1996.
http://mosquitonet.stanford.edu/sigops96/papers/mogul.ps.

[19] Venkata N. Padmanabhan and Jeffrey C. Mogul. Using predictive prefetching to improve world wide web latency. *Computer Communication Review*, 26(3):22–36, 1996.
http://daedalus.cs.berkeley.edu/publications/ccr-july96.ps.gz.

[20] Luigi Rizzo and Lorenzo Vicisano. Replacement policies for a proxy cache.
http://www.iet.unipi.it/~luigi/lrv98.ps.gz, January 1998. Technical Report, University of Pisa.

[21] N. Young. On line file caching. In *Proc. 9th ACM-SIAM Symposium on Discrete Algorithms*. ACM-SIAM, 1998.