Exploiting Regularities in Web Traffic Patterns for Cache Replacement*

Edith Cohen
AT&T Labs–Research
180 Park Avenue, Florham Park, NJ 07932
edith@research.att.com

Haim Kaplan
Tel-Aviv University
Tel-Aviv 69978, Israel
haimk@math.tau.ac.il

May 18, 2002

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 caching, 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 of 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 caching model which we call the *interval caching model*. 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

*An extended abstract of this paper appeared in the Proceedings of the 31st Annual ACM Symposium on Theory of Computing,1999 [8].

in the web environment exhibit different characteristics than in traditional operating systems caches. 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, 15, 23, 22].

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.

A large proxy cache typically serves a large group of users accessing 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 [14] 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 (or minimizes) 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 and the optimal offline 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 show in this paper that when the cache load is steady or predictable then relatively simple cache replacement algorithms are nearly optimal.

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 one can predict a weight function such that after pricing the time according to this function the cache is in steady state.

A second phenomenon we address is patterns of requests to individual pages. Reference locality between accesses to different pages [16, 6, 12] was observed in web traffic [11, 21, 5]. Studies [7, 22] which examined the distribution of lengths of inter-request time intervals between requests for the same page confirm high correlation between time passed since the 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 page. 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 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 until the next request for the requested page. This model[1] was first considered by Lund et al. [19] in the context of holding time policies for virtual circuits (for routing IP traffic over ATM [17, 18]) and was termed IP-paging. The same model was proposed later on for web caching [20, 10]. Following Lund et al. [19], our "yard-stick" is the optimal online algorithm which makes best use of knowing the underlying probability space, but does not know the actual sequence drawn from it apriori. This algorithm is well defined by Markov decision theory for discrete distributions, but is exponential in the number of requests and the descriptions of the distributions [19]. 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.[2]

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. The main difference between the interval caching model and the standard caching model is that in the interval caching model there is no fixed limit on the cache size. Cache replacement algorithms have analogous interval-caching algorithms, whose actions are consistent with the eviction function of the replacement policy. Predictable load can be incorporated in the analogous interval-caching algorithms, and result in decisions that mimic the cache replacement algorithms. For example, an interval-caching algorithm 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 means that items spend approximately the same amount of time in the cache, the analogous interval caching algorithm with the right caching window simulates LRU with high accuracy.

Interval-caching is important for experimental comparison and evaluation of replacement algorithms when only *partial traces*, which contain requests made to a subset of the pages, are available. Instead of simulating a cache replacement algorithm, which requires the full workload of the cache, one can simulate the analogous interval caching algorithm. Simulating the interval caching algorithm is possible using only a partial trace, and in case the load on the cache is indeed steady, both simulations should produce similar results. One example of such experiment is the evaluation of server-assisted cache replacement [10]. 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

[1]with uniform fetching costs and sizes.

[2]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*.

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. [18, 17] to evaluate holding time policies for IP over ATM. In this paper we provide the theoretical foundations for interval-caching evaluation. We formulate necessary conditions under which the cache replacement algorithm and its interval caching analogous produce similar profit-cost tradeoffs.

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 the next request time of the requested page is provided. When page sizes and fetching costs are uniform, this information suffices to perform optimally, by simply evicting the page to be requested furthest in the future [4]. However, when fetching costs and sizes vary, this information is insufficient[3] [15]. 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. For arbitrary fetching costs and sizes we propose an algorithm that is nearly optimal if the load can be predicted correctly.

Our framework yields a near-optimal algorithm for the more general and more realistic IRD caching problem. The optimal strategy for this problem is hard to find even if fetching costs and sizes are uniform [19]. Here is a simple example demonstrating 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.

The rest of the paper is organized as follows. Section 2 describes the basic framework and defines steady state and the interval caching model. Section 3 substantiates the relation between the

[3]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 [15], but an approximate solution can be obtained by rounding the LP-relaxation [9, 1].

interval caching model and the standard caching model for LRU and LFU. Sections 4 and 5 consider the setup where the next request time of the requested page is provided. Section 4 demonstrates our technique in the simple setting where fetching costs and sizes are uniform, and section 5 generalize the results to arbitrary fetching costs and sizes. In Section 6 we extend our results to the IRD caching problem. To simplify the presentation, we prove our results under a steady state assumption on the cache replacement algorithm in question. In section 7 we indicate how to relax the steady-state assumption and extend our results to a situation where the load is not steady but predictable. Section 8 provides experimental evidence to the steady state and predictable load assumptions for LRU, LFU, and the optimal (for uniform fetching costs and sizes) cache replacement algorithm. We performed our experiments using a large proxy log.

2 Preliminaries

Our input is a sequence of requests for web pages. We use the following notation. A request is denoted by r, the requested page by $u(r)$, the time in which r arrives by $t(r)$, the size of $u(r)$ by $s(r)$, and the fetching cost of $u(r)$ by $f(r)$. We also denote by $\delta(r)$ the difference between $t(r')$ and $t(r)$ where r' is the next request for $u(r)$ following r. Additional information about future requests may be given with each request (such as the time interval $\delta(r)$ until the next request for $u(r)$, or a probability distribution on the time of the next request for $u(r)$).

The standard caching model which we shall refer to as the *fixed caching model*, assumes a cache of fixed size k. A *cache replacement algorithm* processes the sequence of requests and for each request for a page that is not in the cache decides whether to put the page in the cache[4]. 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 request is a *hit*. The algorithm gains a *profit* for each hit that is equal to the fetching cost of the requested page[5]. The *profit* of the algorithm on a sequence is the sum of the profits that the algorithm gained through requests to cached items. In case the algorithm A is *randomized* then the profit of A is a random variable. When we refer to the *profit* of a randomized algorithm our intention is to the expected value of this random variable. The profit of an algorithm A is a function of the request sequence s and the cache size k which we consider as a parameter of the cache replacement algorithm. We denote the profit of algorithm A 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 $c_A(s,k) = k * T$. 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 this is indeed the case for all standard cache replacement algorithms.

[4]We use non-forced caching: The cache replacement algorithm can decide not to cache a page.

[5]We shall refer to the fetching cost of a page also as the *profit* associated with the page.

2.1 Eviction function

All standard cache replacement algorithms evict the page that maximizes the value of a particular function. For a cache replacement algorithm A we call this function the *eviction function* of A, and denote it by f_A. The value of the eviction function for one particular page, u, may change with time so the arguments of f_A are a page u and a time t. For example, the eviction function of LRU, $f_{\mathtt{LRU}}(u,t)$ is the time difference between t and the last request to u prior to time t. The variant of LFU that we use in this paper uses the function $f_{\mathtt{LFU}}(u,t)$ whose value is the number of requests to u up to time t divided by t.

Algorithms that use different eviction functions may turn out be equivalent in the sense that on every sequence they take the same decisions. For example, the algorithm LRU′ which uses an eviction function whose value for a page u at the time t is the number of requests since the last request for u prior to time t is equivalent to LRU[6].

We assume in this paper that f_A is defined for all pages no matter whether they are cached at time t or not. We also assume that $f_A(u,t)$ depends only on the requests to u prior to time t. In other words, given t, and the requests to u prior to t then one can calculate $f_A(u,t)$[7].

We say that a sequence s is *nonredundant* with respect to a replacement algorithm A, and an eviction function $f_A(u,t)$, if no two different pages have the same eviction function value at the same time. On nonredundant sequences the actions of A are uniquely determined by the eviction function. For LRU, a sequence is non-redundant if no two different requests arrive at the same time. For LFU, a sequence is nonredundant if in every prefix of the sequence no two different pages have the same number of occurrences.

We shall assume throughout the paper that the input sequence is non-redundant with respect to the cache replacement algorithm and the eviction function in question. This assumption is not too restrictive since if non-redundancy does not hold small perturbations in the parameters of the requests will turn the sequence to be nonredundant. For LFU, for example, small perturbations of the "request count" of the first request for each page u (to be $1 \pm \epsilon_u$) will turn the sequence to be nonredundant.

2.2 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 of the function $e(t) = f_A(u_{\max}(t), t)$ captures the actions of the replacement policy and the load on the cache. When there is a high load on the cache $e(t)$ would be low and vice versa. The knowledge of $e(t)$ allows to determine the time interval during which a page $u(r)$ would be cached without knowing the requests that follow r: Item $u(r)$ remains cached for the maximum Δ such that for every $t \le \Delta$, $f_A(u(r), t(r)+t) \le e(t(r)+t)$.

[6] Assume that we perturb arrival times slightly such that no two pages arrive exactly at the same time.

[7] In fact, with the exception of LFU, for any other cache replacement algorithm A that we consider in this paper $f_A(u,t)$ depends only on the last request for u and the information provided with it.

In reality, *exact* future values of the threshold function $f_A(u_{\max}(t), t)$ are hard to predict, but as we show in Section 8 regularity of traffic patterns suggest that $f_A(u_{\max}(t), t)$ is known *approximately*.

The following definition gives one rather simple model of regularity in traffic patterns. We say that a cache replacement algorithm A with cache of size k is in *ϵ-steady state* for a request sequence s, if there exists a threshold $C(s,k)$ 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 $f_A(u,t) > C(s,k)$.

2. If A evicts a page u at time t then $f_A(u,t) > C(s,k)$.

3. For every t and for every page u' such that the cardinality of the set of pages $\{u | f_A(u,t) < f_A(u',t)\}$ is strictly less than $(1-\epsilon)k$ [8], the inequality $f_A(u',t) \leq C(s,k)$ holds.

If page sizes vary 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

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

we require that $f(u',t) \leq C(s,k)$. Our definition does not allow to evict pages of size greater than ϵk in steady state. If u is evicted then by the second condition right before the eviction $f_A(u,t) > C(s,k)$. However if $s(u) > \epsilon k$ and u attains the maximum eviction value then according to the modified third condition $f_A(u,t) \leq C(s,k)$, which is a contradiction. Since we evict pages only in order to make room for a new ones and pages of size greater than ϵk are never evicted it follows that the cache is always $(1-\epsilon)k$ full in steady state.

From our application point of view it would be important that the cache replacement algorithm is in steady state for a range of cache sizes, and most of the request sequences. Furthermore, thresholds for various cache sizes should be such that $C(s,k)$ is nondecreasing in k. When an algorithm A can be characterized by more than one eviction function we shall prefer a choice that facilitates steady state. In section 7 we discuss how to model caches whose load is predictable but they are not in steady state.

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.

2.3 Interval Caching

We define an *interval caching algorithm* B 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 $\Delta_B(r)$ for which it caches $u(r)$. The page $u(r)$ remains in the cache either until $\Delta_B(r)$ expires, or until it is

[8] To simplify the presentation we assume that $(1-\epsilon)k$ is an integer here and throughout the paper.

requested again by a subsequent request r' (then a new value $\Delta_B(r')$ is generated by the algorithm for the page $u(r') = u(r)$).

If a page is requested again before the time interval allocated to it in its previous request expires, then this is a *hit* and the algorithm gains the profit associated with the page. The *profit* of the algorithm is the sum of the profits it gains from individual requests. The *cost* which the algorithm pays for a request r is $s(r) * c(r)$ where $c(r) = \min\{\Delta_B(r), t(r') - t(r)\}$ and r' is the next request for $u(r)$. The overall *cost* of the algorithm for serving the sequence is the sum of the costs which the algorithm pays to serve individual requests. In case B is randomized then $\Delta_B(r)$ may be drawn according to some probability distribution $L_r(\Delta)$. The profit and the cost of the algorithm are then the expected values (over the random choices of the algorithm) of the corresponding random variables. We denote the profit and the cost of an interval caching algorithm B on a sequence s by $p_B(s)$ and $c_B(s)$ respectively.

We emphasis that the interval caching model in contrast with the fixed caching model does not involve a cache capacity constraint. Therefore, an interval caching algorithm does not deal with the question of which pages to evict, and is not parameterized by cache capacity. A page simply remains in the cache until its caching interval expires. However, to establish a correspondence between the interval caching model and the fixed caching model we parameterize the interval caching model as follows. We define an *interval caching scheme* $\mathcal{B}$ to be an interval caching algorithm that decides on a caching interval, $\Delta_{\mathcal{B}}(r) \equiv \Delta_{\mathcal{B}}(r, V)$, for $u(r)$, based on an external parameter $V \geq 0$ (in addition to the other information it has) subject to the *monotonicity requirement* that $\Delta_{\mathcal{B}}(r, V) \leq \Delta_{\mathcal{B}}(r, V')$ for $V \leq V'$. Note that V is one parameter identical for all the requests in the sequence.

The monotonicity requirement follows from our intention that the parameter V will be analogous to the cache size in the fixed caching model. It follows from the monotonicity requirement that the cost and the profit of an interval caching scheme are nondecreasing functions of V. Intuitively, a particular interval caching scheme and an input sequence carves out a curve in the cost vs. profit plane, and the parameter V specifies a point along this curve. We denote the cost and the profit of an interval caching scheme $\mathcal{B}$ serving a sequence s with parameter V by $c_{\mathcal{B}}(s, V)$ and $p_{\mathcal{B}}(s, V)$, respectively. As an example of how an interval caching scheme may use the parameter V consider a scheme that caches every page $u(r)$ for $\Delta(r, V) = V/s(r)$ seconds.

One can think of an interval caching scheme as a family of interval caching algorithms, one for each value of V. For an interval caching scheme $\mathcal{B}$ we denote the interval caching algorithm corresponding to a fixed value of the parameter V by $\mathcal{B}(V)$. Each algorithm $\mathcal{B}(V)$ can take its caching decisions based on the particular value of V, and the whole collection $\{\mathcal{B}(V)\}$ must satisfy the monotonicity requirement. One can also consider each interval caching algorithm as an interval caching scheme which takes the same decisions for every V.

2.4 Optimality

The optimal cache replacement algorithm, which we denote by `OPT` is the one that generates maximum profit for any given sequence and cache size (including algorithms that know the whole sequence in advance). When fetching costs and sizes are all the same then Belady [4] proved that the optimal algorithm would place a page in the cache only if its next request time is smaller than the next request time of a page which is currently cached. The evicted page in this case is the one with the largest next request time.

We say that an interval caching algorithm B is *optimal* if no interval caching algorithm B' satisfies that $p_{B'}(s) > p_B(s)$ and $c_{B'}(s) \leq c_B(s)$ for some s. Given some cost value C and a sequence s we can find a (possibly randomized) interval caching algorithm that generates maximum profit on s for cost at most C by solving a linear program. If we restrict ourselves to deterministic interval caching algorithms and pages have different sizes and fetching costs then the problem of finding the set of pages that we need to cache to get maximum profit for some fixed cost is NP-complete[9].

It is natural to define an interval caching scheme $\mathcal{B}$ to be *optimal* if for every request sequence s and an interval caching algorithm B' (that may belong to an interval caching scheme) there exists a parameter V such that $p_{\mathcal{B}}(s, V) \geq p_{B'}(s)$ and $c_{\mathcal{B}}(s, V) \leq c_{B'}(s)$. The monotonicity requirement that $\mathcal{B}$ satisfies implies that for each V the corresponding interval caching algorithm in the scheme $\mathcal{B}$ is optimal. However, an optimal interval caching scheme need not exist, since for some sequences the caching decisions of the optimal interval caching algorithms for increasing cost values are not monotone. (We recall that for $\mathcal{B}$ to be an interval caching scheme $\Delta_{\mathcal{B}}(r, V)$ must be nondecreasing with V).

To circumvent this problem we consider a weaker notion of optimality which we call *convex optimality*. To define convex optimality we need the following definition of a convex domination relation among sets of points in the plane. We say that a set $\mathcal{P}_1$ of points in the plane *convexly dominates* another set of points $\mathcal{P}_2$ if every point in $\mathcal{P}_2$ is underneath the upper envelope of $\mathcal{P}_1$. Formally, $\mathcal{P}_1$ *convexly dominates* $\mathcal{P}_2$ if for every point $(c_2, p_2) \in \mathcal{P}_2$ there exists two points $(c_1, p_1), (c_1', p_1') \in P_1$, and $0 \leq \alpha \leq 1$ such that $c_2 = (1-\alpha)c_1 + \alpha c_1'$ and $p_2 \leq (1-\alpha)p_1 + \alpha p_1'$. It is easy to see that the convex domination relation among sets is transitive. An interval caching scheme $\mathcal{B}$ is *convex optimal* if the following two conditions hold.

1. For every s the set of points $\{(c_{\mathcal{B}}(s, V), p_{\mathcal{B}}(s, V))\}$ convexly dominates the set $\{(c_{B'}(s), p_{B'}(s)) \mid B'$ is an arbitrary interval caching algorithm$\}$.
2. All the points in the set $\{(c_{\mathcal{B}}(s, V), p_{\mathcal{B}}(s, V))\}$ are on the upper envelope of this set, and furthermore this is a minimal set defining this upper envelope. (No point in this set is a convex combination of the others.)

We remark that the followings follow from the definition of a convex optimal interval caching scheme.

[9] This is the problem called Knapsack which is known to be NP-complete [13]

1. The set of points $\{(c_{\mathcal{B}}(s,V), p_{\mathcal{B}}(s,V))\}$ generated by an optimal interval caching scheme is unique.

2. If $\mathcal{B}$ is convex optimal then every interval caching algorithm $\mathcal{B}(V)$ is optimal.

The following lemma bounds the performance of the optimal cache replacement algorithm by the performance of an optimal interval caching scheme.

Lemma 2.1 *Consider a request sequence s whose elapsed time is T. Let $\mathtt{OPT}_I$ be a convex optimal interval caching scheme and $\mathtt{OPT}$ the optimal cache replacement algorithm. Fix a parameter $V \geq 0$ Then, for cache size $k \leq \lfloor c_{\mathtt{OPT}_I}(s,V)/T \rfloor$ we have that $p_{\mathtt{OPT}}(s,k) \leq p_{\mathtt{OPT}_I}(s,V)$.*

Proof. It is easy to see that OPT is deterministic. We can deduce from the replacement sequence of OPT with cache size k an interval caching algorithm that for every sequence s, generates profit $p_{\mathtt{OPT}}(s,k)$ for a cost no greater than $k * T$. This interval caching algorithm caches $u(r)$ for an interval of length $\delta(r)$ if and only if OPT with cache of size k does.

From the convex optimality of $\mathtt{OPT}_I$ follows that there are thresholds V_1 and V_2 such that $k * T = (1-\alpha)c_{\mathtt{OPT}_I}(s,V_1) + \alpha c_{\mathtt{OPT}_I}(s,V_2)$ and $p_{\mathtt{OPT}}(s,k) \leq (1-\alpha)p_{\mathtt{OPT}_I}(s,V_1) + \alpha p_{\mathtt{OPT}_I}(s,V_2)$.

If $p_{\mathtt{OPT}}(s,k) > p_{\mathtt{OPT}_I}(s,V)$ we obtain that $p_{\mathtt{OPT}_I}(s,V) \leq (1-\alpha)p_{\mathtt{OPT}_I}(s,V_1) + \alpha p_{\mathtt{OPT}_I}(s,V_2)$. which contradicts the concavity of the profit as a function of the cost since $k * T \leq c_{\mathtt{OPT}_I}(s,V)$ and the cost and profit are monotonically increasing with V. ■

We say that an interval caching scheme A_I and a replacement policy A are *analogous* if $\Delta_{A_I}(r,V)$ is the maximum Δ such that $f_A(u(r), t(r)+t) \leq V$ for all $t \leq \Delta$, or infinity if no such t exists. The value $f_A(u(r), t(r)+t)$ is calculated assuming that no additional requests to $u(r)$ occur after time $t(r)$. For every cache replacement algorithm A that we consider we give an analogous interval caching scheme A_I that generates similar profit-cost trade-offs to A on sequences where A is in steady state.

3 Interval caching counterparts for LRU and LFU

In this section we consider LRU and LFU and show that they generate similar profit to cost tradeoffs as their interval caching analogous when they are in steady state. Both LRU and LFU share the property that the pages in the cache at any one time are the k smallest[10] with respect to the value of the eviction function. We call a cache replacement algorithm with this property *monotone*. We show that every monotone cache replacement algorithm A and its interval caching analogous A_I exhibit similar profit-cost tradeoffs on sequences where A is in steady state.

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.

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

Theorem 3.1 *Let A be a monotone cache replacement algorithm and let A_I be its analogous interval-caching algorithm. If A with cache size k is in ϵ-steady state on a sequence s then*

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

where T is the elapsed time of s and $C(s,k)$ is the steady state threshold.

Proof. Consider a run of A_I on a sequence s with parameter $C(s,k)$. At any given time, the cache of A_I contains only pages cached by A with cache size k. Furthermore the total size of pages cached by A_I is $(1-\epsilon)k$. Hence, the cost bounds follow.

The profit bounds follow from the fact that A_I caches all pages cached by A with cache size $(1-\epsilon)k$. Hence, the profit of A_I is at least the profit of A with cache size $(1-\epsilon)k$ and at most the profit of A with cache size k. ■

The interval caching algorithm $\texttt{LRU}_I$ that corresponds to LRU is the algorithm that caches every request for an interval of length V. From Theorem 3.1 we obtain that when LRU is in steady state, its profit to cost trade-offs are similar to those of $\texttt{LRU}_I$.

We use a normalized eviction function for LFU: We define $f_{\texttt{LFU}}(u,t)$ to be the number of requests for u issued so far, divided by t. The corresponding interval caching algorithm, $\texttt{LFU}_I$ caches u for an interval of length Δ, where Δ is maximum such that the number of occurrences of u seen so far divided by $t+\Delta$ is still no less than V. Theorem 3.1 implies that when LFU is in steady state, LFU and $\texttt{LFU}_I$ have similar performance.

4 Optimal interval-caching and replacement policies for uniform sizes and costs.

The current section serves as a "warm up" for Sections 5 and 6. Our technique of using the optimal interval-caching policy ($\texttt{OPT}_I$ defined below) to derive a near-optimal replacement policy (namely, $\texttt{OPT}'$ defined below) in steady state is illustrated here in a simple set up. We assume here that fetching costs and sizes are all equal to one.

As mentioned in Section 2.4 when fetching costs and sizes are all the same the optimal algorithm OPT evicts the page whose next request time is furthest in the future. The interval caching policy $\texttt{OPT}_I$ that is analogous to OPT caches a page u until its next request if and only if the interrequest time is less than V. In fact $\texttt{OPT}_I$ is also analogous to a cache replacement algorithm whose eviction value for a page u at any time t is the length of the time interval between the last request to u prior to time t and the first request to u after time t. We call this cache replacement algorithm $\texttt{OPT}'$. The value of the eviction function $f_{\texttt{OPT}'}$ remains constant between consecutive requests for page u in contrast with the eviction function $f_{\texttt{OPT}}$ that linearly decreases between consecutive requests for u.

The following theorem establishes that among deterministic interval caching algorithms $\texttt{OPT}_I$ is an optimal interval caching scheme.

Theorem 4.1 *Consider a sequence s such that $\delta(r) \neq \delta(r')$ for every $r \neq r'$ in s. Let $c_B(s)$ and $p_B(s)$ be the cost and profit, respectively, incurred by some deterministic interval caching algorithm B on the sequence s. Then, for some V, $p_{\mathtt{OPT}_I}(s,V) = p_B(s)$ and $c_{\mathtt{OPT}_I}(s,V) \leq c_B(s)$.*

Proof. Since B is deterministic $p_B(s)$ is an integer. Since $\delta(r) \neq \delta(r')$ for every $r \neq r'$ in s it follows from the definition of $\mathtt{OPT}_I$ that it obtains every possible integer profit value for some V with minimum cost. ■

If we consider also randomized algorithms then $\mathtt{OPT}_I$ is a convex optimal scheme as stated by the following theorem. This theorem is a corollary of Theorem 5.1 that we prove in Section 5.

Theorem 4.2 *When fetching costs and sizes are all the same $\mathtt{OPT}_I$ is a convex optimal interval caching scheme.*

Generally, the hit rate of $\mathtt{OPT}'$ can be arbitrarily worse than the hit rate of $\mathtt{OPT}$. Consider as an example a cache of size 1, a page u requested on times $0, 1, 2, \ldots$ and a set of pages $u_0, \ldots, u_n$ where each u_i is requested twice, in times $i - i(i+1)\delta/2$ and $i + 1 - i(i+3)\delta/2$. The inter-request interval of page u_i is $1 - i\delta$ and page u_i first arrives just before page u_{i-1} is requested again. When processing this sequence $\mathtt{OPT}'$ never caches u and evicts each u_i just before it is requested again. Hence, it incurs only a single cache hit on page u_n. $\mathtt{OPT}$, in contrast, keeps u in cache and incurs n cache hits.

We will show that when $\mathtt{OPT}'$ is in steady state then the cache replacement algorithms $\mathtt{OPT}$, $\mathtt{OPT}'$ and the interval-caching algorithm $\mathtt{OPT}_I$ have similar profit-cost trade-offs. In contrast to $\mathtt{LRU}$ and $\mathtt{LFU}$, the value of the eviction function of $\mathtt{OPT}$ and $\mathtt{OPT}'$ can decrease as well as increase when we get a new request for u. As a consequence, it is not necessarily true that the pages in the cache are the k smallest with respect to the eviction function and hence, the proof of Theorem 3.1 is not applicable. We do, however, prove the analogous claim:

Theorem 4.3 *Assume that $\mathtt{OPT}'$ with cache size k is in ϵ-steady state for a sequence s. Then,*

$$\begin{array}{rcccl}(1-\epsilon)c_{\mathtt{OPT}'}(s,k) \equiv (1-\epsilon)k * T & \leq & c_{\mathtt{OPT}_I}(s, C(s,k)) & \leq & c_{\mathtt{OPT}'}(s,k) \equiv k * T \\ p_{\mathtt{OPT}}(s,(1-\epsilon)k) & \leq & p_{\mathtt{OPT}_I}(s, C(s,k)) & \leq & p_{\mathtt{OPT}'}(s,k)\ ,\end{array}$$

where T is the elapsed time of the request sequence, and $C(s,k)$ is the threshold of the steady-state.

Proof. Consider a run of $\mathtt{OPT}_I$ on a sequence s with parameter $V = C(s,k)$. The cost bounds follow from the fact that at any given time, the cache of $\mathtt{OPT}_I$ contains a subset (or all) pages cached by $\mathtt{OPT}'$ with cache size k and at least $(1-\epsilon)$ of the pages cached by $\mathtt{OPT}'$ with cache size k.

The upper bound on profit follows from the fact that pages cached by $\mathtt{OPT}_I$ are always a subset of pages cached by $\mathtt{OPT}'$ with cache size k.

Since $\mathtt{OPT}_I$ is a convex optimal interval caching algorithm, Lemma 2.1 implies that any cache replacement algorithm with cache size smaller than $\lfloor c_{\mathtt{OPT}_I}(s, C(s,k))/T \rfloor$ cannot have more profit than $p_{\mathtt{OPT}_I}(s, C(s,k))$. Since we proved that $(1-\epsilon)k \leq \lfloor c_{\mathtt{OPT}_I}(s, C(s,k))/T \rfloor$ the statement

$$p_{\mathtt{OPT}}(s,(1-\epsilon)k) \leq p_{\mathtt{OPT}_I}(s, C(s,k))$$

now follows. ■

The following lemma is a consequence of optimality of OPT.

Lemma 4.1 OPT *is such that for all* s *and* k, $(1-\epsilon)p_{\mathtt{OPT}}(s,k) \le p_{\mathtt{OPT}}(s,(1-\epsilon)k)$. ■

Proof. Consider the cache as consisting of k slots, and associate each hit with the slot containing the page for which the hit is obtained. Consider the best $(1-\epsilon)k$ slots. Clearly the number of hits accumulated by these slots is at least $(1-\epsilon)p_{\mathtt{OPT}}(s,k)$. Therefore there is an algorithm with profit at least $(1-\epsilon)p_{\mathtt{OPT}}(s,k)$ and cache of size $(1-\epsilon)k$. ■

Remark: The lemma holds when pages have varying fetching costs, but does not necessarily hold when pages have varying sizes. It is also false for OPT′, LRU, and LFU. To see that this relation does not always hold for OPT′, consider the sequences in the example above demonstrating the suboptimality of OPT′ with cache sizes $k=2$ and $\epsilon=0.5$ (therefore $(1-\epsilon)k=1$). It is also easy to construct sequences on which OPT and OPT′ perform the same actions and hence, the inequality does hold. We obtain that the quantities $(1-\epsilon)p_{\mathtt{OPT}'}(s,k)$ and $p_{\mathtt{OPT}'}(s,(1-\epsilon)k)$ are incomparable. ■

Combining Theorem 4.3 and Lemma 4.1 we obtain the following corollary showing that when OPT′ is in steady state then both OPT′ and $\mathtt{OPT}_I$ have similar profits. The profit of these two algorithms when OPT′ is in steady state is close to the profit of OPT.

Corollary 4.1 *Under the assumptions of Theorem 4.3 we have* $(1-\epsilon)p_{\mathtt{OPT}'}(s,k) \le (1-\epsilon)p_{\mathtt{OPT}}(s,k) \le p_{\mathtt{OPT}_I}(s,C(s,k)) \le p_{\mathtt{OPT}'}(s,k)$.

Proof. The first inequality follows from the fact that OPT is an optimal cache replacement algorithm. The second inequality follows from Lemma 4.1 and Theorem 4.3, and the third follows from Theorem 4.3. ■

5 Caching files with varying sizes and fetching costs

In this section we consider caching when pages have varying sizes and costs. We consider a cache replacement algorithm GOPT and its analogous interval caching scheme $\mathtt{GOPT}_I$. Both GOPT and $\mathtt{GOPT}_I$ assume that as each request r arrives, the algorithm is also told the time, $\delta(r)$, until the next request for page $u(r)$. We first establish the optimality of $\mathtt{GOPT}_I$ and then use it to show that GOPT is nearly optimal when it is in ϵ-steady state for some small epsilon.

When $\delta(r)$ is known we may assume that the only choices of an interval caching algorithm B upon receiving a request r is either to cache $u(r)$ for $\delta(r)$ time or not to cache it at all. Let r be a request for a page $u(r)$. The profit of caching $u(r)$ for time length $\delta(r)$ following r is $\mathrm{profit}(r)=f(r)$, and the corresponding cost is $\mathrm{cost}(r)=s(r)\delta(r)$. Let B be an interval caching algorithm, and let L_r be the probability that B caches $u(r)$ for time $\delta(r)$ following r. Then the profit of B from r is $\mathrm{profit}_B(r) \equiv L_r \mathrm{profit}(r)$ and the cost of B from r is $\mathrm{cost}_B(r) \equiv L_r \mathrm{cost}(r)$. The profit of B on the whole sequence is the sum of the profits of B on the requests in s, i.e. $\mathrm{profit}_B(s) = \sum_{r\in s} \mathrm{profit}_B(r)$. Similarly, the cost of B on s is $\mathrm{cost}_B(s) = \sum_{r\in s} \mathrm{cost}_B(r)$.

5.1 Convex optimal interval caching algorithm

Define the utility of a request r to be $\text{util}(r) = \frac{\text{profit}(r)}{\text{cost}(r)}$. The interval caching scheme $\texttt{GOPT}_I$ with a threshold V caches $u(r)$ for $\delta(r)$ with probability 1 if and only if $\frac{1}{\text{util}(r)} \leq V$ (or in other words if and only if $\text{util}(r) \geq 1/V$). Clearly $\Delta_{\texttt{GOPT}_I}(r, V)$ is nondecreasing with V; it equals zero up to $V = 1/\text{util}(r)$, and it equals $\delta(r)$ for every other V. We also observe that $\text{profit}_{\texttt{GOPT}_I}(s, V)$ and $\text{cost}_{\texttt{GOPT}_I}(s, V)$ are nondecreasing step functions of V for every s, and the number of different pairs $(\text{profit}_{\texttt{GOPT}_I}(s, V), \text{cost}_{\texttt{GOPT}_I}(s, V))$ for any s is no greater than the number of requests. By the definition of $\texttt{GOPT}_I$ if $\text{cost}_{\texttt{GOPT}_I}(s, V) = \text{cost}_{\texttt{GOPT}_I}(s, V')$ then $\text{profit}_{\texttt{GOPT}_I}(s, V) = \text{profit}_{\texttt{GOPT}_I}(s, V')$. Therefore we can define a function $f(x)$ by setting $\text{profit}_{\texttt{GOPT}_I}(s, V) = f(\text{cost}_{\texttt{GOPT}_I}(s, V))$ and linearly interpolating so it is defined for all x. It also follows from the definition of $\texttt{GOPT}_I$ that f is concave since if $\text{cost}_{\texttt{GOPT}_I}(s, V) < \text{cost}_{\texttt{GOPT}_I}(s, V')$ then $\frac{\text{profit}_{\texttt{GOPT}_I}(s,V)}{\text{cost}_{\texttt{GOPT}_I}(s,V)} > \frac{\text{profit}_{\texttt{GOPT}_I}(s,V')}{\text{cost}_{\texttt{GOPT}_I}(s,V')}$.

The following theorem establishes that $\texttt{GOPT}_I$ yields optimal trade-offs of profit and cost.

Theorem 5.1 *Algorithm* $\texttt{GOPT}_I$ *is convex optimal interval caching scheme.*

Proof. It is clear from the reasoning above that $\texttt{GOPT}_I$ satisfies the second condition of the definition of convex optimal interval caching scheme. To prove that $\texttt{GOPT}_I$ satisfies the first condition of being convex optimal it is enough to show that $(c_B(s), p_B(s))$ is underneath the upper envelope of the profit-cost pairs generated by $\texttt{GOPT}_I$ for every interval caching algorithm B.

Fix an interval caching algorithm B. Let V_1 be the maximum threshold such that $c_{\texttt{GOPT}_I}(s, V_1) \leq c_B(s)$ and let V_2 be the minimum threshold such that $c_{\texttt{GOPT}_I}(s, V_2) \geq c_B(s)$. Note that $V_1 \leq V_2$, and both are well defined (to see this notice that the cost and profit of each request are finite positive numbers, $\texttt{GOPT}_I$ with threshold value $V = 0$ does not cache any request and $\texttt{GOPT}_I$ with threshold $V = \infty$ caches all requests r for which $\Delta(r) \neq \infty$).

Let S_1 and S_2 be the sets of requests cached by $\texttt{GOPT}_I$ with thresholds V_1 and V_2, respectively, and let $D = S_2 - S_1$. By the definitions of $\texttt{GOPT}_I$, S_1, and S_2, all the requests in D have the same utility u, where $1/V_2 \leq u \leq 1/V_1$. Let $r_1, \ldots, r_n$ be the requests in s ordered by decreasing utility values and let j be the smallest such that $\sum_{l=1}^{j} \text{cost}_B(r_l) \geq \text{cost}_{\texttt{GOPT}_I}(s, V_1)$. Define $\rho \equiv \frac{\sum_{l=1}^{j} \text{cost}_B(r_l) - \text{cost}_{\texttt{GOPT}_I}(s,V_1)}{\text{cost}(r_j)}$ and define $u' \equiv \frac{\rho\text{profit}_B(r_j) + \sum_{l=j+1}^{n} \text{profit}_B(r_l)}{\text{cost}_B(s) - \text{cost}_{\texttt{GOPT}_I}(s,V_1)}$. By the definition of $\texttt{GOPT}_I$, $\text{cost}_{\texttt{GOPT}_I}(s, V_1)$, is the total cost of requests with utility larger than u. Therefore, $u' \leq u$. Choose $0 \leq \alpha \leq 1$ such that

$$\text{cost}_B(s) - \text{cost}_{\texttt{GOPT}_I}(s, V_1) = \alpha(\text{cost}_{\texttt{GOPT}_I}(s, V_2) - \text{cost}_{\texttt{GOPT}_I}(s, V_1))$$

By the definition of u' we know that

$$\text{profit}_B(s) - \text{profit}_{\texttt{GOPT}_I}(s, V_1) \leq u'(\text{cost}_B(s) - \text{cost}_{\texttt{GOPT}_I}(s, V_1))$$

Combining the previous two inequalities and using the fact that $u' \leq u$ we obtain that

$$\text{profit}_B(s) - \text{profit}_{\texttt{GOPT}_I}(s, V_1) \leq \alpha(\text{profit}_{\texttt{GOPT}_I}(s, V_2) - \text{profit}_{\texttt{GOPT}_I}(s, V_1))$$

as required. ■

5.2 The cache replacement algorithm GOPT

We define the utility of page u to be the utility of the most recent request for u. Algorithm GOPT is defined as follows. If a cache miss occurs when requesting a page $u(r)$, algorithm GOPT checks whether it can evict a set of pages with inverse-utility value larger than the inverse-utility of $u(r)$ so as to make room for $u(r)$ in the cache. If indeed that is the case $\mathtt{GOPT}_I$ evicts those pages and caches $u(r)$ instead. Algorithm GOPT generalizes OPT′ for the settings where pages have varying sizes and fetching costs. As for OPT′, GOPT has the property that the eviction value of a page u is fixed between requests for u. Clearly the interval caching scheme $\mathtt{GOPT}_I$ is analogous to GOPT.

Remark: Another algorithm that uses the utilities of different pages in order to take decisions is the LANDLORD algorithm of Young [23] (see also [7]). As GOPT also LANDLORD evicts pages in the cache with minimum utility. However LANDLORD after every eviction adjusts the utility of all pages in the cache. After evicting a page with utility x, LANDLORD decrease the utility of all other pages in the cache by a factor of x.

The following theorem is a generalization of Theorem 4.3. It shows that GOPT generates almost as much profit as OPT when it is in steady state. It also implies that GOPT and $\mathtt{GOPT}_I$ with the appropriate parameter generate similar costs and profits when GOPT is in steady state.

Theorem 5.2 *Assume that* GOPT *is in ϵ-steady state for a sequence s and cache size k, with threshold $C = C(s,k)$. Then,*

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

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

Proof. Consider a run of $\mathtt{GOPT}_I$ on a sequence s with parameter C. The upper bound on the cost follows from the fact that at any given time, the cache of $\mathtt{GOPT}_I$ contains a subset (or all) of the pages cached by GOPT with cache size k. The lower bound on the cost follows from the fact that at any time $\mathtt{GOPT}_I$ caches pages that are cached by GOPT of total size no less than $(1-\epsilon)k$. The upper bound on the profit follows from the fact that pages cached by $\mathtt{GOPT}_I$ are always a subset of pages cached by GOPT with cache size k.

To prove the lower bound on the profit consider OPT with cache size k, and construct an interval caching algorithm B as in the proof of Lemma 2.1 such that $p_B(s) = p_{\mathtt{OPT}}(s,k)$ and $c_B(s) \leq k*T$. If $c_B(s) \leq c_{\mathtt{GOPT}_I}(s,C)$ then from Theorem 5.1 follows that $p_{\mathtt{GOPT}_I}(s,C) \geq p_B(s) = p_{\mathtt{OPT}}(s,k)$ and the lower bound follows. So assume $c_B(s) > c_{\mathtt{GOPT}_I}(s,C)$.

Sort the requests cached by B by decreasing utility values. Let $r_1, r_2, \ldots r_n$ be the sorted sequence of requests and let j be the minimum such that $\sum_{l=1}^{j} \mathrm{cost}_B(r_l) \geq \mathrm{cost}_{\mathtt{GOPT}_I}(s,C)$. Define $\rho = \frac{\sum_{l=1}^{j} \mathrm{cost}_B(r_l) - c_{\mathtt{GOPT}_I}(s,C)}{\mathrm{cost}_B(r_j)}$. By the definition of $\mathtt{GOPT}_I$ we have that

$$\sum_{l=1}^{j-1} \text{profit}_B(r_l) + (1-\rho)\text{profit}_B(r_j) \le p_{\mathtt{GOPT}_I}(s,C) \ ,$$

and

$$\frac{\rho\,\text{profit}_B(r_j) + \sum_{l=j+1}^{n} \text{profit}_B(r_l)}{c_B(s) - c_{\mathtt{GOPT}_I}(s,C)} \le \frac{1}{C} \ .$$

Combining the last two inequalities and using the facts that $p_B(s) = p_{\mathtt{OPT}}(s,k)$ and $c_B(s) \le k * T$ we obtain that

$$p_{\mathtt{OPT}}(s,k) - p_{\mathtt{GOPT}_I}(s,C) \le \frac{k * T - c_{\mathtt{GOPT}_I}(s,C)}{C} \ . \tag{1}$$

Since $\mathtt{GOPT}_I$ caches to completion all the requests r such that $\text{cost}(r)/\text{profit}(r) \le C$ we have that

$$p_{\mathtt{GOPT}_I}(s,C) \ge c_{\mathtt{GOPT}_I}(s,C)/C \ . \tag{2}$$

By combining Equations (1) and (2) we obtain that

$$\frac{p_{\mathtt{OPT}}(s,k) - p_{\mathtt{GOPT}_I}(s,C)}{p_{\mathtt{GOPT}_I}(s,C)} \le \frac{k * T - c_{\mathtt{GOPT}_I}(s,C)}{c_{\mathtt{GOPT}_I}(s,C)} \ ,$$

and thus,

$$p_{\mathtt{GOPT}_I}(s,C) \ge p_{\mathtt{OPT}}(s,k)\frac{c_{\mathtt{GOPT}_I}(s,C)}{k * T} \ge (1-\epsilon)p_{\mathtt{OPT}}(s,k) \ .$$

The lower bound $p_{\mathtt{OPT}}(s,(1-\epsilon)k) \le p_{\mathtt{GOPT}_I}(s,C)$ follows from the lower bound on $c_{\mathtt{GOPT}_I}(s,C)$, Theorem 5.1, and Lemma 2.1. ■

Since `OPT` is the optimal cache replacement algorithm its profit is larger than the profit of `GOPT` so we have the following corollary that shows that `GOPT` and $\mathtt{GOPT}_I$ generate similar profits (and similar costs as was already established in Theorem 5.2) when `GOPT` is in steady state.

Corollary 5.1 *Under the conditions of Theorem 5.2 we have that* $(1-\epsilon)p_{\mathtt{GOPT}}(s,k) \le p_{\mathtt{GOPT}_I}(s,C) \le p_{\mathtt{GOPT}}(s,k)$, *and* $p_{\mathtt{GOPT}}(s,(1-\epsilon)k) \le p_{\mathtt{GOPT}_I}(s,C)$.

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 [19]. 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) [10]. 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 of the time to the next request for the same page.

In this section we assume that each request r arrives with a cumulative distribution function $N_r(\delta)$ (or alternatively a density function $n_r(\delta)$), and the time to the next request of $u(r)$ is drawn according to this distribution. We assume that the underlying probability distribution on request sequences is determined ahead of time so subsequent requests do not depend on actions taken by the algorithm. Furthermore, we also assume that $N_r(\delta)$ does not depend on requests for pages other than $u(r)$ that follow r. To simplify the presentation we impose the following additional assumptions.

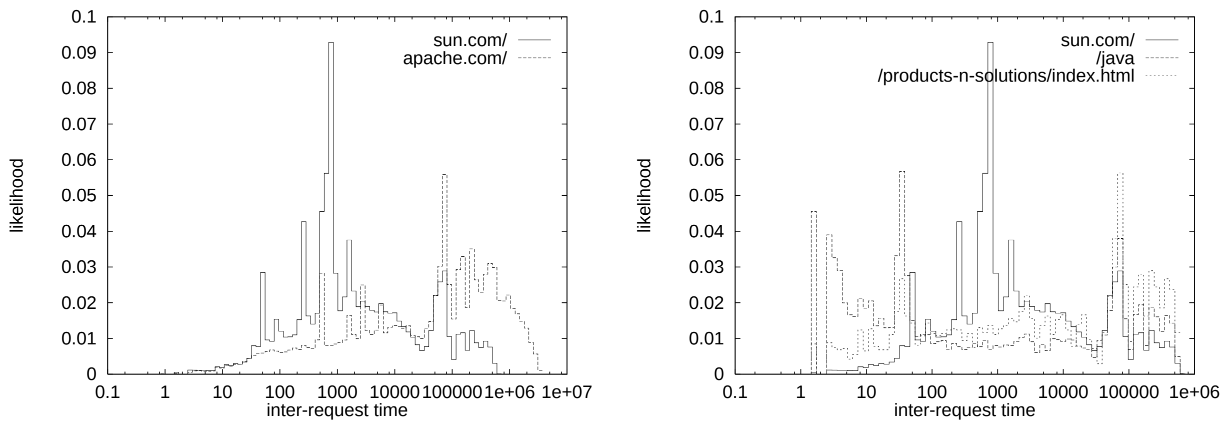
1. The elapsed time of any possible input sequence is bounded by $T_{\max}$. First request arrives at time $t = 0$.
2. Time is discrete so requests can arrive only at a finite set of points $t_0 = 0, t_1, \ldots, t_n = T_{\max}$. This implies that each cumulative distribution function $N_r(\delta)$ is piecewise linear and may change only at $t_j - t(r)$, for $t_j > t(r)$.
3. We consider $1 - N_r(T_{\max} - t(r))$ as the probability that there is no request for $u(r)$ subsequent to r.
4. The total number of different requests is finite. We denote by R the set of all possible requests.

We denote any distribution on request sequences satisfying the conditions above $\overline{s}$. Each such distribution is a possible input for our algorithms. We denote by T the average over $\overline{s}$ of the elapsed time of a request sequence.

We measure a cache replacement algorithm A working with cache size k by its *expected profit*, $p_A(\overline{s}, k) = E(p_A(s, k))$, over $\overline{s}$[11]. Since the average elapsed time of a sequence is T, the *expected cost* of serving a sequence drawn from $\overline{s}$ is $k * T$. The *optimal* cache replacement algorithm in these settings, denoted by `OPT` in this section, is the algorithm which maximizes the expected profit over $\overline{s}$ among all algorithms that know $\overline{s}$, but do not know actual request sequence drawn from $\overline{s}$. In this sense, `LRU` (Least Recently Used) is optimal when the probability density functions $n_r(\delta)$ are identical for all requests and are non-increasing with time (therefore, it is always more profitable to keep a more recently-requested page), and `LFU` (Least Frequently Used) is optimal when each page is requested with a fixed frequency (assuming uniform fetching costs and sizes).

Studies [7, 22] which examined lengths of inter-request time intervals for pages in various proxy logs show strong locality of reference (high correlation between time passed since last request and likelihood of another request), which supports the use of `LRU`. Further experiments (see for example Figure 1) show that per-page inter-request times distributions vary considerably between web sites and within a site. These distributions differ in form and in their parameters such as mean or median. This variability supports the use of `LFU` which considers previous per-page history, and suggests that better results can be obtained by incorporating per-page inter-request distributions in a finer manner in the replacement policy.

[11] In case A is randomized then for any fixed s, $p_A(s, k)$ is the *expected* profit over the random choices of A.



(a) apache.com and sun.com homepages (b) various pages on the sum.com web site

Figure 1: Inter-request histogram for pages (URL's) using web-server logs from the apache.com and sun.com web sites.

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 [19]. 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. We start by describing an interval caching scheme $\texttt{MPG}_I$ that generates optimal tradeoffs in the interval caching model. Then we introduce the 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.

6.1 The interval caching model

In the current probabilistic setting we measure an interval caching scheme $\mathcal{B}$ with a threshold V by its *expected profit*, $p_{\mathcal{B}}(\overline{s}, V) = E(p_{\mathcal{B}}(s, V))$, and its *expected cost*, $c_{\mathcal{B}}(\overline{s}, V) = E(c_{\mathcal{B}}(s, V))$, over the probability distribution $\overline{s}$ of request sequences.

Let R be the set of all requests occurring in $\overline{s}$ (Recall that each such r consists of a page $u(r)$, the time $t(r)$ in which it occurs, and a distribution $N_r(\delta)$ of the time to the next request of $u(r)$). We associate a probability $q(r)$ with each request $r \in R$ that is the sum of the probabilities of the sequences which contain r. If $\mathcal{B}$ is an interval caching scheme then it is possible to summarize the actions of $\mathcal{B}$ by mapping each $r \in R$ to a probability density function $L_{r,\mathcal{B},V}(\Delta)$ that captures the likelihood (over $\overline{s}$ and the random choices of the algorithm) that upon encountering a request r, $\mathcal{B}$ with parameter V caches $u(r)$ for time period Δ. We omit the subscript V from L and other functions defined below when B is an interval caching algorithm. We omit other subscripts when they are clear from the context. Our assumptions on $\overline{s}$ imply that we may assume that $L_r(\Delta)$ is positive only for $\Delta = t_i - t(r)$ where i is such that $t_i > t(r)$. Analogously to our monotonicity requirement of $\Delta_{\mathcal{B}}(r, V)$ in Section 2 we require that $\sum_{t>t_i} L_{r,V}(t_i - t(r))$ is nondecreasing function of V for every i and r.

For a request $r \in R$ we denote by $\text{profit}_r(\Delta)$ the expected profit (over $\overline{s}$) of caching $u(r)$ for an interval of length Δ following r. Clearly $\text{profit}_r(\Delta) = N_r(\Delta)$. We also denote the expected cost of caching $u(r)$ for an interval of length Δ following r by $\text{cost}_r(\Delta)$. It is easy to see that $\text{cost}_r(\Delta) = \int_0^\Delta (1 - N_r(\delta))d\delta$ since in case the next request for $u(r)$ following r is a hit we charge r only for the caching period up to the hit. Let $t(r) = t_j$, for some $1 \leq j \leq n$. By linearity of expectation the profit of $\mathcal{B}$ from a request r is $\text{profit}_{\mathcal{B},V}(r) = \sum_{i=j+1}^{n} L_{r,V}(t_i - t(r))\text{profit}_r(t_i - t(r))$ and the expected cost is $\text{cost}_{\mathcal{B},V}(r) = \sum_{i=j+1}^{n} L_{r,V}(t_i - t(r))\text{cost}_r(t_i - t(r))$. Summing over the requests we obtain that the expected profit of an interval caching scheme $\mathcal{B}$ is

$$p_{\mathcal{B}}(\overline{s}, V) = \sum_{r\in R} q(r)\text{profit}_{\mathcal{B},V}(r)$$

and the expected cost is

$$c_{\mathcal{B}}(\overline{s}, V) = \sum_{r\in R} q(r)\text{cost}_{\mathcal{B},V}(r) \ .$$

Analogously to Section 2.4 we say that an interval caching scheme $\mathcal{B}$ is *convex optimal* if the following two conditions hold

1. For every $\overline{s}$ the set of points $\{(c_{\mathcal{B}}(\overline{s}, V), p_{\mathcal{B}}(\overline{s}, V))\}$ convexly dominates the set $\{(c_{B'}(\overline{s}), p_{B'}(\overline{s})) \mid B'$ is an arbitrary interval caching algorithm$\}$.

2. All the points in the set $\{(c_{\mathcal{B}}(\overline{s}, V), p_{\mathcal{B}}(\overline{s}, V))\}$ are on the upper envelope of this set. No point $(c_{\mathcal{B}}(\overline{s}, V'), p_{\mathcal{B}}(\overline{s}, V'))$ is a convex combination of other points in $\{(c_{\mathcal{B}}(\overline{s}, V), p_{\mathcal{B}}(\overline{s}, V))\}$.

It is clear from this definition that the set $\{(c_{\mathcal{B}}(\overline{s}, V), p_{\mathcal{B}}(\overline{s}, V))\}$ generated by any convexly optimal interval caching scheme $\mathcal{B}$ is unique.

The following lemma is analogous to Lemma 2.1. In the proof of this lemma we use the assumption that for every request r the distribution $N_r(\delta)$ is independent of forthcoming requests to other pages.

Lemma 6.1 *Let $\texttt{OPT}_I$ be a convex optimal interval caching algorithm, and let T be the average elapsed time of the request sequences. Fix a threshold value $V \geq 0$. Then, for cache size $k \leq \lfloor c_{\texttt{OPT}_I}(\overline{s}, V)/T \rfloor$ we have that $p_{\texttt{OPT}}(\overline{s}, k) \leq p_{\texttt{OPT}_I}(\overline{s}, V)$, where $\texttt{OPT}$ is the optimal cache replacement algorithm in the fixed caching model.*

Proof. The proof is analogous to the proof of Lemma 2.1. However since here our input is a probability distribution on request sequences it is not clear any more that the optimal cache replacement policy is deterministic. This makes the argument which constructs form the optimal cache replacement algorithm an interval caching algorithm with the same profit and cost more subtle. In the following we show how to construct from the replacement sequences of OPT on $\overline{s}$ an interval caching algorithm B whose expected profit is $p_{\texttt{OPT}}(\overline{s}, k)$ and its expected cost is no greater than $k * T$.

We split the profit of OPT among the requests r by associating a hit to a page u, with the last request for u before the hit. We also split the cost of OPT among the requests by charging to r the cost of caching $u(r)$ from r to the next request for $u(r)$ or until it is evicted. We construct the interval caching algorithm B such that its profit and its cost from a request r is the same as the profit and the cost of OPT that is associated with r, respectively.

Let r be a request in $\overline{s}$, and let s be a sequence of requests ending with r that is a prefix of some sequence in $\overline{s}$. Fix a sequence $s' = (r'_1, r'_2, r'_3, \ldots, r'_n)$ of requests to pages other than $u(r)$ that follow s in some sequence of $\overline{s}$. Let ss' denote all sequences of requests that start with s and continue as s' with respect to pages other than $u(r)$. Clearly different members of ss' differ only by their requests to $u(r)$ that follow r. Let $q(ss')$ be the probability that a sequence of requests drawn from $\overline{s}$ starts with s and continues as s' with respect to pages other than $u(r)$. We define a probability distribution on ss' such that for every $z \in ss'$, $p(z)$ is the conditional probability that a sequence drawn from $\overline{s}$ is z given that the sequence is in ss'. In the rest of the proof we assume that sequences are drawn from ss' according to this probability distribution.

We define B by defining its actions separately for every r. For every possible pair of sequences s, s' where s ends with r and s' does not contain requests for $u(r)$ we construct a function $L_r^{ss'}(\Delta)$. The function $L_r^{ss'}(\Delta)$ is defined such that if an interval caching algorithm upon receiving r, caches $u(r)$ for a period Δ with probability $L_r^{ss'}(\Delta)$ then its profit and its cost from r on sequences drawn from ss' is the same as the profit and the cost of OPT associated with r on sequences from ss'. Once we have defined the functions $L_r^{ss'}(\Delta)$ for every possible ss' we define $l_{r,B}(\Delta)$ as the convex combination $\sum_{ss'} q(ss') L_r^{ss'}(\Delta)$. Using the assumption that the next request for $u(r)$ is drawn according to N_r regardless of requests to pages other than $u(r)$ that follow r, it follows that the profit and the cost of B are equal to the profit and the cost of OPT from r. In the rest of the proof we describe how to define $L_r^{ss'}(\Delta)$.

We define probabilities $p_0, p_1, \ldots, p_n$ based on the probability distribution defined on ss' and the random choices of OPT (in case it is randomized) as follows. Let p_0 be the probability that OPT while serving a sequence drawn from ss', decides to cache $u(r)$. Let p_i be the probability that while serving a sequence drawn from ss', $u(r)$ stays in the cache of OPT after $t(r'_i)$ and the request for $u(r)$ following r does not arrive before $t(r'_i)$. Let $\Delta_i = t(r'_i) - t(r)$, for $1 \le i \le n$, and $\Delta_{\max} = T_{\max} - t(r)$. We recall our assumption that $N_r(\delta)$ does not depend on requests to pages other than $u(r)$ that follow r, and the assumption that OPT does not know the actual sequence drawn. It follows from these assumptions that the expected profit of OPT from caching $u(r)$ after serving r on sequences drawn from ss' is

$$\begin{aligned} p_0 * \mathrm{profit}_r(\Delta_1) \quad &+ \quad p_1 * \frac{\mathrm{profit}_r(\Delta_2) - \mathrm{profit}_r(\Delta_1)}{(1 - N_r(\Delta_1))} \quad + \quad \cdots \\ &+ \quad p_i * \frac{\mathrm{profit}_r(\Delta_{i+1}) - \mathrm{profit}_r(\Delta_i)}{(1 - N_r(\Delta_i))} \quad + \quad \cdots \quad + \quad p_n * \frac{\mathrm{profit}_r(\Delta_{\max}) - \mathrm{profit}_r(\Delta_n)}{(1 - N_r(\Delta_n))} \; , \end{aligned} \tag{3}$$

and similarly its cost is

$$p_0 * \mathrm{cost}_r(\Delta_1) \quad + \quad p_1 * \tfrac{\mathrm{cost}_r(\Delta_2)-\mathrm{cost}_r(\Delta_1)}{(1-N_r(\Delta_1))} \quad + \quad \cdots \quad + \quad p_i * \tfrac{\mathrm{cost}_r(\Delta_{i+1})-\mathrm{cost}_r(\Delta_i)}{(1-N_r(\Delta_i))} \quad + \quad \cdots \quad + \quad p_n * \tfrac{\mathrm{cost}_r(\Delta_{\max})-\mathrm{cost}_r(\Delta_n)}{(1-N_r(\Delta_n))}\ . \tag{4}$$

Now we define $L_r^{ss'}(\Delta_i) = q_{i-1}$, $L_r^{ss'}(\Delta_{\max}) = q_n$, and for every other Δ, $L_r^{ss'}(\Delta) = 0$, where $q_0, q_2, \ldots, q_n$ is the probability distribution that solves the following linear system of equations

$$\begin{aligned}
q_0 + q_1 + q_2 + q_3 + \cdots + q_n &= p_0 \\
(1 - N_r(\Delta_1))(q_1 + q_2 + q_3 + \cdots + q_n) &= p_1 \\
(1 - N_r(\Delta_2))(q_2 + \cdots + q_n) &= p_2 \\
\cdots &= \cdots \\
\cdots &= \cdots \\
(1 - N_r(\Delta_n)) * (q_n) &= p_n
\end{aligned}$$

From the definition of $q_0, \ldots, q_n$ follows that if an interval caching algorithm cache $u(r)$ according to $L_r^{ss'}$ then its profit and cost from r is the same as that of OPT on sequences from ss' as described in equations (3) and (4). ■

6.2 The interval-caching algorithm $\mathtt{MPG}_I$

We define

$$\mathrm{util}_r(\Delta_1, \Delta_2) \equiv \frac{\mathrm{profit}_r(\Delta_2) - \mathrm{profit}_r(\Delta_1)}{\mathrm{cost}_r(\Delta_2) - \mathrm{cost}_r(\Delta_1)} \equiv \frac{N_r(\Delta_2) - N_r(\Delta_1)}{\int_{\Delta_1}^{\Delta_2}(1 - N_r(\delta))d\delta}\ ,$$

that is the expected profit gain of increasing the caching period of $u(r)$ from Δ_1 to Δ_2 divided by the expected cost of this increase. (We omit the subscript r when it is clear from the context.) Algorithm $\mathtt{MPG}_I$ with parameter V caches $u(r)$ for an interval of length $\Delta_{\mathtt{MPG}_I}(r, V)$ where $\Delta_{\mathtt{MPG}_I}(r, V) = \max\{z \mid \forall y < z,\ \ \frac{1}{\mathrm{util}_r(y,z)} \leq V\}$. That is, $\Delta_{\mathtt{MPG}_I}(r, V)$ is the maximum point such that for every $y < \Delta_{\mathtt{MPG}_I}(r, V)$, the expected profit of increasing the caching period from y to $\Delta_{\mathtt{MPG}_I}(r, V)$ divided by the expected cost of this increase is no less than $1/V$. In terms of the general definitions at the beginning of this section, the function $L_{r,\mathcal{B},V}(\Delta)$ when $\mathcal{B} = \mathtt{MPG}_I$ is one for $\Delta = \Delta_{\mathtt{MPG}_I}(r, V)$, and zero for every other Δ.

Remark: Keshav et al. [17] 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 $\mathcal{B}$ as the total time items are cached plus the total cost of cache misses. It is straightforward to check that given a probability distribution on request sequences algorithm $\mathtt{MPG}_I(\mathcal{O})$ minimizes the overall cost among all algorithms that know the probability distribution. Hence $\mathtt{MPG}_I$ is identical to the *adaptive policy* of [17]. $\mathtt{MPG}_I$ was also used in [10] to conduct experiments in a setup where the request sequence is not completely known. ■

Given a distribution function $N_r(\delta)$ such that $N_r(T_{\max} - t(r))$ is positive, we define intervals $I_1^r = [0, \Delta_1^r]$, $I_2^r = (\Delta_1^r, \Delta_2^r]$, $\ldots$,$I_{m_r}^r = (\Delta_{m_r-1}^r, \Delta_{m_r}^r]$ as follows. The point Δ_1^r is the maximum $0 < z \le T_{\max} - t(r)$ that maximizes the quantity

$$\text{util}(0, z) = \frac{N_r(z)}{\int_0^z (1 - N_r(\delta))d\delta} \ .$$

Assuming we have defined $\Delta_1^r, \ldots \Delta_{i-1}^r$ if

$$\text{util}(\Delta_{i-1}^r, z) = \frac{N_r(z) - N_r(\Delta_{i-1}^r)}{\int_{\Delta_{i-1}^r}^z (1 - N_r(\delta))d\delta} \ ,$$

is nonzero for some $\Delta_{i-1}^r < z \le T_{\max} - t(r)$ then Δ_i^r $(i > 1)$ is the maximum $z > \Delta_{i-1}^r$ that maximizes $\text{util}(\Delta_{i-1}^r, z)$. We define $\text{util}(I_i^r) \equiv \text{util}(\Delta_{i-1}^r, \Delta_i^r)$. It follows from this definition that $\text{util}(I_{i-1}^r) > \text{util}(I_i^r)$ for every i. The following lemma shows that $\Delta_{\texttt{MPG}_I}(r, V)$ always takes one of the values $0 \equiv \Delta_0^r, \Delta_1^r, \ldots, \Delta_{m_r}^r$ for every V.

Lemma 6.2 *An equivalent alternative definition for the choice of algorithm* $\texttt{MPG}_I$ *is to cache* $u(r)$ *for an interval* $[0, \Delta_i^r]$ *for the maximum* i *such that* $\text{util}(I_i^r) \ge 1/V$.

Proof. By the definition of Δ_{i+1}^r and the choice of i we have $\text{util}(\Delta_i^r, z) < 1/V$ for every $z > \Delta_i^r$. To conclude we claim that for all $y < \Delta_i^r$, $\text{util}(y, \Delta_i^r) \ge 1/V$. Let j be such that $\Delta_{j-1}^r \le y < \Delta_j^r$ for some $j \le i$. By the definition of Δ_j^r, $\text{util}(\Delta_{j-1}^r, y) \le \text{util}(\Delta_{j-1}^r, \Delta_j^r)$ so $\text{util}(y, \Delta_j^r) \ge \text{util}(\Delta_{j-1}^r, \Delta_j^r) \ge 1/V$. Since $\text{util}(y, \Delta_i^r) \ge \min\{\text{util}(y, \Delta_j^r), \text{util}(I_{j+1}^r), \ldots, \text{util}(I_i^r)\}$, and $\text{util}(I_{i'}^r) \ge \text{util}(I_i^r)$ for all $i' \le i$ the claim follows. ∎

We note that the set of values $\{\text{cost}_{\texttt{MPG}_I}(\overline{s}, V) \,|\, V \ge 0\}$ and the set of values $\{\text{profit}_{\texttt{MPG}_I}(\overline{s}, V) \,|\, V \ge 0\}$ are finite. For the definition of $\texttt{MPG}_I$ follows that if $\text{cost}_{\texttt{MPG}_I}(\overline{s}, V) = \text{cost}_{\texttt{MPG}_I}(\overline{s}, V')$ for some $V' > V$ then $\text{profit}_{\texttt{MPG}_I}(\overline{s}, V) = \text{profit}_{\texttt{MPG}_I}(\overline{s}, V')$. Therefore we can define a function $f(x)$ by setting $\text{profit}_{\texttt{MPG}_I}(\overline{s}, V) = f(\text{cost}_{\texttt{MPG}_I}(\overline{s}, V))$ and extending it continuously for every x by a linear interpolation. It also follows from the definition of $\texttt{MPG}_I$ that if $\text{cost}_{\texttt{MPG}_I}(\overline{s}, V) < \text{cost}_{\texttt{MPG}_I}(\overline{s}, V')$ for some $V' > V$ then the ratio $\text{profit}_{\texttt{MPG}_I}(\overline{s}, V)/\text{cost}_{\texttt{MPG}_I}(\overline{s}, V)$ is strictly larger than the ratio $\text{profit}_{\texttt{MPG}_I}(\overline{s}, V')/\text{cost}_{\texttt{MPG}_I}(\overline{s}, V')$. Therefore f is concave and the scheme $\texttt{MPG}_I$ satisfies the second requirement of a convex optimal interval caching scheme. We next prove that $\texttt{MPG}_I$ also satisfies the first requirement of a convex optimal interval caching scheme and thereby establish that $\texttt{MPG}_I$ is a convex optimal interval caching scheme.

Fix a distribution $\overline{s}$. We say that a family of interval caching algorithms F_1 *convexly dominates* a family of interval caching algorithms F_2 on $\overline{s}$ if the set $\{(c_B(\overline{s}), p_B(\overline{s})) \,|\, B \in F_1\}$ convexly dominates the set $\{(c_B(\overline{s}), p_B(\overline{s})) \,|\, B \in F_2\}$. Our first lemma towards establishing the convex optimality of $\texttt{MPG}_I$ is the following.

Lemma 6.3 *The family of interval caching algorithms* F_2 *such that for every* $B \in F_2$ *the function* $L_{r,B}(\Delta)$ *is nonzero only on* $0, \Delta_1^r, \Delta_2^r, \ldots, \Delta_{m_r}^r$ *for every* r *convexly dominates all interval caching algorithms on* $\overline{s}$.

Proof. Let $\mathcal{D}_m$ be the family of all interval caching algorithms for which the number of points Δ such that for some r, $L_r(\Delta)$ is nonzero and $\Delta \notin \{0, \Delta_1^r, \Delta_2^r, \ldots, \Delta_{m_r}^r\}$ is m. We show that $\mathcal{D}_m$ convexly dominates $\mathcal{D}_{m+1}$ from which the theorem follows.

Let D be an interval caching algorithm in $\mathcal{D}_{m+1}$ Let Δ be a point such that $L_{r,D}(\Delta)$ is nonzero, and $\Delta \notin \{0, \Delta_1^r, \Delta_2^r, \ldots, \Delta_{m_r}^r\}$. Assume $\Delta \in (\Delta_i^r, \Delta_{i+1}^r]$. Let D_1 be an algorithm that behaves exactly like D except that $L_{r,D_1}(\Delta_i^r) = L_{r,D}(\Delta_i^r) + L_{r,D}(\Delta)$ and $L_{r,D_1}(\Delta) = 0$. Let D_2 be an algorithm that behaves exactly like D except that $L_{r,D_2}(\Delta_{i+1}^r) = L_{r,D}(\Delta_{i+1}^r) + L_{r,D}(\Delta)$ and $L_{r,D_2}(\Delta) = 0$. Let $\alpha = \frac{\mathrm{cost}_r(\Delta) - \mathrm{cost}_r(\Delta_i^r)}{\mathrm{cost}_r(\Delta_{i+1}^r) - \mathrm{cost}_r(\Delta_i^r)}$.

It is easy to check that $c_D(\overline{s}) = (1-\alpha)c_{D_1}(\overline{s}) + \alpha c_{D_2}(\overline{s})$. Furthermore, since by the definition of Δ_{i+1}^r, $\mathrm{util}(\Delta_i^r, \Delta) \leq \mathrm{util}(\Delta_i^r, \Delta_{i+1}^r)$ we obtain that

$$\begin{aligned}
\mathrm{profit}_D(\overline{s}) - \mathrm{profit}_{D_1}(\overline{s}) &= L_{r,D}(\Delta)(\mathrm{profit}_r(\Delta) - \mathrm{profit}_r(\Delta_i^r)) \\
&= L_{r,D}(\Delta)\mathrm{util}_r(\Delta_i^r, \Delta)(\mathrm{cost}_r(\Delta) - \mathrm{cost}_r(\Delta_i^r)) \\
&= L_{r,D}(\Delta)\mathrm{util}_r(\Delta_i^r, \Delta)\alpha(\mathrm{cost}_r(\Delta_{i+1}^r) - \mathrm{cost}_r(\Delta_i^r)) \\
&\leq L_{r,D}(\Delta)\mathrm{util}_r(\Delta_i^r, \Delta_{i+1})\alpha(\mathrm{cost}_r(\Delta_{i+1}^r) - \mathrm{cost}_r(\Delta_i^r)) \\
&= L_{r,D}(\Delta)\alpha(\mathrm{profit}_r(\Delta_{i+1}^r) - \mathrm{profit}_r(\Delta_i^r)) \\
&= \alpha(\mathrm{profit}_{D_2}(\overline{s}) - \mathrm{profit}_{D_1}(\overline{s})) \ .
\end{aligned}$$

Since D_1 and D_2 are in $\mathcal{D}_m$ this implies that $\mathcal{D}_m$ convexly dominates $\mathcal{D}_{m+1}$ ■

Lemma 6.4 *The family of interval caching algorithms F_1 such that for every $B \in F_1$ the function $L_{r,B}(\Delta)$ is nonzero only on* one of $0, \Delta_1^r, \Delta_2^r, \ldots, \Delta_{m_r}^r$ *for every r, convexly dominates all interval caching algorithms.*

Proof. By Lemma 6.3, and the transitivity of the convex domination relation, it is enough to show that the set of interval caching algorithms F_1 convexly dominates the set F_2 where for every $B \in F_2$ $L_{r,B}(\Delta)$ is nonzero only on $0, \Delta_1^r, \Delta_2^r, \ldots, \Delta_{m_r}^r$ for every r.

Let $\mathcal{D}_m \subseteq F_2$ be the set of all algorithms $D \in F_2$ with m pairs Δ_i^r, Δ_j^r such that both $L_{r,D}(\Delta_i^r)$ and $L_{r,D}(\Delta_j^r)$ are nonzero and for $i < l < j$, $L_{r,D}(\Delta_l^r) = 0$. We prove that $\mathcal{D}_m$ convexly dominates $\mathcal{D}_{m+1}$ from which the theorem follows.

Let B be an algorithm in $\mathcal{D}_{m+1}$. Let Δ_i^r, Δ_j^r be a pair such that both $L_{r,B}(\Delta_i^r)$ and $L_{r,B}(\Delta_j^r)$ are nonzero, and $L_{r,B}(\Delta_l^r) = 0$ for every $i \leq l \leq j$. Let $y = L_{r,B}(\Delta_i^r) + L_{r,B}(\Delta_j^r)$, and let B_1 be an algorithm identical to B except that $L_{r,B_1}(\Delta_j^r) = y$ and $L_{r,B_1}(\Delta_i^r) = 0$. Similarly let B_2 be an algorithm identical to B except that $L_{r,B_2}(\Delta_i^r) = y$ and $L_{r,B_2}(\Delta_j^r) = 0$. Let $\alpha = L_{r,B}(\Delta_i^r)/y$. From the definitions of B_1 and B_2 follow that $c_B(\overline{s}) = (1-\alpha)c_{B_1}(\overline{s}) + \alpha c_{B_2}(\overline{s})$ and $p_B(\overline{s}) = (1-\alpha)p_{B_1}(\overline{s}) + \alpha p_{B_2}(\overline{s})$. Since B_1 and B_2 belong to $\mathcal{D}_m$ we obtain that $\mathcal{D}_m$ convexly dominates $\mathcal{D}_{m+1}$. ■

Using Lemma 6.4 we can now establish that $\mathtt{MPG}_I$ is convexly optimal.

Theorem 6.1 *Algorithm $\mathtt{MPG}_I$ is a convex optimal interval caching scheme.*

Proof. We already observed that $\texttt{MPG}_I$ satisfies the second requirement of an optimal interval caching scheme. To prove that it also satisfies the first we note that by Lemma 6.4 we have to show that the algorithms $\texttt{MPG}_I(V)$ convexly dominates the set of algorithms F_1 that make a single choice of caching time with probability one upon encountering every request r, and the choice is one of $0, \Delta_1^r, \Delta_2^r, \ldots, \Delta_{m_r}^r$.

From the distribution $\overline{s}$ we construct a single sequence s as follows. Recall that for an $r \in \overline{s}$, $q(r)$ is the probability of encoutering r in a request sequence drawn from $\overline{s}$. We associate with r pages $u_1, u_2, \ldots, u_{m_r}$ and add to s two requests, r_i and r'_i, one time unit apart, to page u_i, for every $1 \leq i \leq m_r$. We define the fetching cost of u_i to be $q(r)(\text{profit}_r(\Delta_i^r) - \text{profit}_r(\Delta_{i-1}^r))$. We define the size of u_i to be $q(r)(\text{cost}_r(\Delta_i^r) - \text{cost}_r(\Delta_{i-1}^r))$.

For every algorithm $B \in F_1$, there exists an algorithm B' serving the single sequence s whose profit and cost are equal to the profit and cost of B. Algorithm B' is defined such that if B when serving request r caches $u(r)$ for time interval $\Delta = \Delta_i^r$ then B' caches for one time unit the pages $u_1, u_2, \ldots, u_i$ when serving $r_1, \ldots, r_i$, respectively. By Theorem 5.1 algorithm $\texttt{GOPT}_I$ is convex optimal and therefore there are V_1 and V_2 such that $c_B(\overline{s}) = c_{B'}(s) = (1-\alpha)c_{\texttt{GOPT}_I}(s, V_1) + \alpha c_{\texttt{GOPT}_I}(s, V_2)$ and $p_B(\overline{s}) = p_{B'}(s) = (1-\alpha)p_{\texttt{GOPT}_I}(s, V_1) + \alpha p_{\texttt{GOPT}_I}(s, V_2)$.

The theorem now follows by observing that the profit and the cost of $\texttt{GOPT}_I(V)$ on s are equal to the profit and the cost of $\texttt{MPG}_I(V)$ on $\overline{s}$. ■

6.3 The cache replacement algorithm MPG

The interval caching algorithm $\texttt{MPG}_I$ is analogous to the cache replacement algorithm MPG that we define as follows. Let u be a page and let r be the last request such that $u = u(r)$. Let t be a time such that $t \geq t(r)$, and define $I(u,t)$ to be the interval among $I_1^r, I_2^r, \ldots, I_{m_r}^r$ that contains $t - t(r)$ if such an interval exists. We define the eviction function of MPG, $f_{\texttt{MPG}}(u,t)$, to be $\frac{1}{\text{util}_r(I(u,t))}$ if $I(u,t)$ is defined and ∞ otherwise. When a request for page u at time t generates a cache miss, algorithm MPG compares $f_{\texttt{MPG}}(u,t)$ with $f_{\texttt{MPG}}(u_{\max}(t), t)$ where $u_{max}(t)$ is the page in the cache which maximizes $f_{\texttt{MPG}}(u,t)$. If $f_{\texttt{MPG}}(u,t) < f_{\texttt{MPG}}(u_{\max}(t), t)$ then page u replaces page $u_{\max}(t)$ 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 it is in steady state, we extend the definition of steady state to our probabilistic setup of this section where the input is a probability distribution on request sequences.

Let $\overline{s}$ be a probability distribution on request sequences and let S a subset of the sequences in $\overline{s}$. For a cache replacement algorithm A we define its *expected profit restricted to* S as

$$p_A(\overline{s}, k)|S = \frac{\sum_{s \in S} q(s) p_A(s,k)}{\sum_{s \in S} q(s)} .$$

We define the *cost of* A *restricted to* S analogously and denote it by $c_A(\overline{s}, k)|S$. For an interval caching scheme A_I we similarly define $p_{A_I}(\overline{s}, V)|S$ and $c_{A_I}(\overline{s}, V)|S$.

We say that $\mathtt{MPG}$ with cache size k is in ϵ,δ-steady state on $\overline{s}$ if there is a set D of sequences in $\overline{s}$ such that

1. $\mathtt{MPG}$ is in ϵ-steady state (according to the definition in Section 2.2) on every sequence of D with the same threshold that we denote by $C(\overline{s},k)$.

2. The following inequalities hold with $C = C(\overline{s},k)$.

$$\begin{array}{rcccc} (1-\delta)c_{\mathtt{MPG}_I}(\overline{s},C) & \leq & c_{\mathtt{MPG}_I}(\overline{s},C)|D & \leq & \frac{c_{\mathtt{MPG}_I}(\overline{s},C)}{1-\delta} \\ (1-\delta)p_{\mathtt{MPG}_I}(\overline{s},C) & \leq & p_{\mathtt{MPG}_I}(\overline{s},C)|D & & \\ (1-\delta)c_{\mathtt{MPG}}(\overline{s},k) & \leq & c_{\mathtt{MPG}}(\overline{s},k)|D & \leq & \frac{c_{\mathtt{MPG}}(\overline{s},k)}{1-\delta} \\ & & p_{\mathtt{MPG}}(\overline{s},k)|D & \leq & \frac{p_{\mathtt{MPG}}(\overline{s},k)}{1-\delta} \end{array} \tag{5}$$

Note that if $\sum_{s\in D} q(s) \geq 1-\delta$ then the upper bounds in Equation (5) follow. It will follow from Theorem (6.2) that steady state conditions imply also that $p_{\mathtt{MPG}}(\overline{s},k)|D$ is not too small compared to $p_{\mathtt{MPG}}(\overline{s},k)$ and that $p_{\mathtt{MPG}_I}(\overline{s},C)|D$ is not too large compared to $p_{\mathtt{MPG}_I}(\overline{s},C)$. In summary, small values of δ mean that the sequences in D dominate the expected cost and expected profit of $\mathtt{MPG}_I$ with threshold C and the expected cost and expected profit of $\mathtt{MPG}$ with cache size k. The following theorem shows that $\mathtt{MPG}$, $\mathtt{OPT}$, and $\mathtt{MPG}_I$ have similar performance when δ and ϵ are small.

Theorem 6.2 *Suppose the replacement policy* $\mathtt{MPG}$ *with cache size* k *is in* ϵ,δ*-steady state on* $\overline{s}$ *with threshold* $C = C(\overline{s},k)$. *Then,*

$$\begin{array}{rcccc} (1-\delta)^2(1-\epsilon)k*T & \leq & c_{\mathtt{MPG}_I}(\overline{s},C) & \leq & \frac{k*T}{(1-\delta)^2} \\ (1-\delta)^2(1-\epsilon)p_{\mathtt{OPT}}(\overline{s},k) & \leq & p_{\mathtt{MPG}_I}(\overline{s},C) & \leq & \frac{p_{\mathtt{MPG}}(\overline{s},k)}{(1-\delta)^2} \\ p_{\mathtt{OPT}}\left((1-\delta)^2(1-\epsilon)k\right) & \leq & p_{\mathtt{MPG}_I}(\overline{s},C) & & \end{array}$$

where T *is the averaged elapsed time of a sequence in* $\overline{s}$.

Proof. Let D be the set of sequence for which $\mathtt{MPG}$ is ϵ,δ-steady state. For each $s \in D$, $\mathtt{MPG}_I$ caches at least $(1-\epsilon)k$ and at most k of the pages cached by $\mathtt{MPG}$ with cache size k throughout the elapsed time of s. Thus if T_s is the elapsed time of s we have

$$(1-\epsilon)k*T_s \;\leq\; c_{\mathtt{MPG}_I}(s,C) \;\leq\; k*T_s\ .$$

Summing for all $s \in D$ we obtain

$$(1-\epsilon)c_{\mathtt{MPG}}(\overline{s},k)|D \;\leq\; c_{\mathtt{MPG}_I}(\overline{s},C)|D \;\leq\; c_{\mathtt{MPG}}(\overline{s},k)|D\ . \tag{6}$$

From the definition of steady state we obtain that $c_{\mathtt{MPG}_I}(\overline{s},C) \leq \frac{c_{\mathtt{MPG}_I}(\overline{s},C)|D}{(1-\delta)}$ and $c_{\mathtt{MPG}}(\overline{s},k)|D \leq \frac{c_{\mathtt{MPG}}(\overline{s},k)}{(1-\delta)}$. Combining these two inequalities with Equation (6) we obtain that

$$c_{\mathtt{MPG}_I}(\overline{s},C) \leq \frac{c_{\mathtt{MPG}}(\overline{s},k)}{(1-\delta)^2}\ . \tag{7}$$

Similarly the steady state assumption implies that $c_{\mathtt{MPG}_I}(\overline{s},C) \geq (1-\delta)(c_{\mathtt{MPG}_I}(\overline{s},C)|D)$ and $c_{\mathtt{MPG}}(\overline{s},k)|D \geq (1-\delta)c_{\mathtt{MPG}}(\overline{s},k)$. Combining these two inequalities with Equation (6) we obtain that

$$c_{\mathtt{MPG}_I}(\overline{s},C) \geq (1-\epsilon)(1-\delta)^2 c_{\mathtt{MPG}}(\overline{s},k) \tag{8}$$

Combining Equations (7) and (8) and the assumption that the averaged elapsed time of a sequence in $\overline{s}$ is T we obtain the bounds on the cost of $\mathtt{MPG}_I$ stated in the Theorem.

To prove the upper and lower bounds on the profit of $\mathtt{MPG}_I$ we note that for each $s \in D$, $\mathtt{MPG}_I$ at any time caches a subset of the pages that $\mathtt{MPG}$ caches. Therefore for $s \in D$

$$p_{\mathtt{MPG}_I}(s,C) \leq p_{\mathtt{MPG}}(s,k)$$

so by summing up these inequalities over D we obtain

$$p_{\mathtt{MPG}_I}(\overline{s},C)|D \leq p_{\mathtt{MPG}}(\overline{s},k)|D \ . \tag{9}$$

The steady state assumption implies that $p_{\mathtt{MPG}_I}(\overline{s},C) \leq \frac{p_{\mathtt{MPG}_I}(\overline{s},C)|D}{(1-\delta)}$ and $p_{\mathtt{MPG}}(\overline{s},k)|D \leq \frac{p_{\mathtt{MPG}}(\overline{s},k)}{(1-\delta)}$. Combining these inequalities with Equation (9) gives the upper bound on the profit stated in the Theorem.

To obtain the first lower bound stated in the theorem consider the interval caching algorithm B which generates the same expected profit and expected cost as $\mathtt{OPT}$ that we constructed in Lemma 6.1. If $c_{\mathtt{MPG}_I}(\overline{s},C)$ is larger than $k*T$ then by the convex optimality of $\mathtt{MPG}$ we obtain that $p_{\mathtt{MPG}_I}(\overline{s},C) \geq p_B(\overline{s}) = p_{\mathtt{OPT}}(\overline{s},k)$ and the lemma follows. So in the following we assume that $c_{\mathtt{MPG}_I}(\overline{s},C) \leq k*T$.

For an interval I_j^r we define $\mathrm{cost}_B(I_j^r) = \sum L_r(t_i - t(r))(\mathrm{cost}_r(t_i - t(r)) - \mathrm{cost}_r(\Delta_{i-1}^r)) + x * (\mathrm{cost}_r(\Delta_i^r) - \mathrm{cost}_r(\Delta_{i-1}^r))$ where the sum is over all t_i such that $\Delta_{i-1}^r < t_i - t(r) \leq \Delta_i^r$ and x is the total probability with which B caches $u(r)$ for an interval of length at least Δ_i^r. Similarly we define $\mathrm{cost}_B(I_j^r) = \sum L_r(t_i - t(r))(\mathrm{profit}_r(t_i - t(r)) - \mathrm{profit}_r(\Delta_{i-1}^r)) + x*(\mathrm{profit}_r(\Delta_i^r) - \mathrm{profit}_r(\Delta_{i-1}^r))$ where the sum is over all t_i such that $\Delta_{i-1}^r < t_i - t(r) \leq \Delta_i^r$ and x is the total probability with which B caches $u(r)$ for an interval of length at least Δ_i^r. Let $\mathcal{I}$ be the set of intervals I_j^r such that $\mathrm{cost}_B(I_j^r) \neq 0$. Sort the intervals in $\mathcal{I}$ by decreasing utility values. Let $I_1,\ldots,I_n$ be these intervals in their sorted order. Let j be the minimum such that $\sum_{l=1}^{j} \mathrm{cost}_B(I_l) \geq \mathrm{cost}_{\mathtt{MPG}_I}(\overline{s},C)$. Define $\rho = \frac{\sum_{l=1}^{j} \mathrm{cost}_B(r_l) - c_{\mathtt{MPG}_I}(\overline{s},C)}{\mathrm{cost}_B(I_j)}$. By the definition of $\mathtt{MPG}_I$ we have that

$$\sum_{l=1}^{j-1} \mathrm{profit}_B(I_l) + (1-\rho)\mathrm{profit}_B(I_j) \leq p_{\mathtt{MPG}_I}(\overline{s},C)$$

and

$$\frac{\rho * \mathrm{profit}_B(I_j) + \sum_{l=j+1}^{n} \mathrm{profit}_B(I_l)}{c_B(\overline{s}) - c_{\mathtt{MPG}_I}(\overline{s},C)} \leq \frac{1}{C}$$

By combining the last two inequalities and the facts that $p_B(\overline{s}) = p_{\mathtt{OPT}}(\overline{s},k)$ and $c_B(\overline{s}) \leq k*T$ we obtain that

$$p_{\mathtt{OPT}}(\overline{s},k) - p_{\mathtt{MPG}_I}(\overline{s},C) \leq \frac{k*T - c_{\mathtt{MPG}_I}(\overline{s},C)}{C} \ . \tag{10}$$

Since $\mathtt{MPG}_I$ caches $u(r)$, with probability one, for the entire interval I_i^r if $\text{util}(I_i^r) \geq 1/C$ and $\text{cost}_{\mathtt{MPG}_I}(I_i^r) = 0$ otherwise. we have that

$$p_{\mathtt{MPG}_I}(\overline{s},C) \geq c_{\mathtt{MPG}_I}(\overline{s},C)/C \ . \tag{11}$$

By combining Equations (10) and (11) we obtain that

$$\frac{p_{\mathtt{OPT}}(\overline{s},k) - p_{\mathtt{MPG}_I}(\overline{s},C)}{p_{\mathtt{MPG}_I}(\overline{s},C)} \leq \frac{k*T - c_{\mathtt{MPG}_I}(\overline{s},C)}{c_{\mathtt{MPG}_I}(\overline{s},C)} \ ,$$

and by using the lower bound on the cost of $\mathtt{MPG}_I$ we obtain that

$$p_{\mathtt{MPG}_I}(\overline{s},C) \geq p_{\mathtt{OPT}}(\overline{s},k)\frac{c_{\mathtt{MPG}_I}(\overline{s},C)}{k*T} \geq (1-\epsilon)(1-\delta)^2 p_{\mathtt{OPT}}(\overline{s},k) \ .$$

Finally, to obtain the second lower bound on the profit of $\mathtt{MPG}_I$ we use the lower bound on the cost together with Lemma 6.1. ■

The following corollary establishes that $\mathtt{MPG}$ is (nearly) optimal when in ϵ,δ-steady state, and ϵ and δ are small. It also show that $\mathtt{MPG}$ and $\mathtt{MPG}_I$ have similar profits when $\mathtt{MPG}$ is in steady state.

Corollary 6.1 *Under the assumption of Theorem 6.2*

$$(1-\delta)^4(1-\epsilon)p_{\mathtt{MPG}}(\overline{s},k) \leq (1-\delta)^4(1-\epsilon)p_{\mathtt{OPT}}(\overline{s},k) \leq (1-\delta)^2 p_{\mathtt{MPG}_I}(\overline{s},C) \leq p_{\mathtt{MPG}}(\overline{s},k) \ .$$

7 Predictable but non-steady load

We proved in previous sections that $\mathtt{OPT}'$, $\mathtt{GOPT}$, and $\mathtt{MPG}$ are nearly optimal when the algorithm is in ϵ-steady state for a small ϵ. Here we extend our results to a situation where the cache replacement algorithm is not in steady state but we can predict the load on the cache, that is, we can predict the function $e(t) = f_A(u_{\max}(t), t)$. We model the *load* on the cache by a weight function on time $\pi(t) \geq 0$, which we refer to as a *price function*. Intuitively, the function π assigns different values to different times, according to $e(t)$. Times of high usage (when evicted pages have low eviction values, i.e. $e(t)$ is low), have higher price and vice versa. We show in this section that all our results generalize in a straightforward manner to the situation where after weighting the time by the price function (time t is mapped to time $\int_0^t \pi(t)dt$) the cache replacement algorithm in question is in steady state.

Our experimental observations in Section 8 show that the load under the cache replacement algorithms that we experimented with is indeed predictable. Even though the algorithms that we simulated were in steady state for $\epsilon \approx 1/5$, the predictability of the load indicates that with a price function these algorithms may be in steady state for a smaller ϵ. The question of what is the best way to calculate a price function that make a particular cache replacement algorithm be in steady state for a small ϵ, is left open for further research.

When we price time by a price function π we have to incorporate π into all the definitions of caching costs of various algorithms throughout the paper. The cost of unit storage during the time interval $[t_1, t_2]$ is $\int_{t_1}^{t_2} \pi(t)dt$ with a price function π rather than $|t_2 - t_1|$ without a price function. For example, the cost of a cache replacement algorithm with cache size k serving a sequence s whose elapsed time is T changes from $k * T$ to $k \int_0^T \pi(t)dt$.

7.1 Incorporating prices into LRU and LFU

With a price function π we consider the algorithm $\mathtt{LRU}^\pi$ which is defined such that $f_{\mathtt{LRU}^\pi}(u,t) = \int_{t(r)}^{t} \pi(t)dt$ where r is the last request to u prior to time t. Similarly we consider $\mathtt{LFU}^\pi$ whose eviction function is $f^\pi_{\mathtt{LFU}}(u,t) = \frac{y_u}{\int_0^t \pi(t)dt}$ where y_u is the number of requests to u before time t[12].

Prices do not affect the decisions taken by LRU and LFU, i.e. LRU and $\mathtt{LRU}^\pi$ cache exactly the same set of pages at any time, and a similar statement holds for LFU and $\mathtt{LFU}^\pi$. This follows since $f_{\mathtt{LRU}}(u,t) \le f_{\mathtt{LRU}}(u',t)$ if and only if $f_{\mathtt{LRU}^\pi}(u,t) \le f_{\mathtt{LRU}^\pi}(u',t)$. The eviction function of $\mathtt{LRU}^\pi$ or $\mathtt{LFU}^\pi$, however may facilitate steady state, when the eviction function of LRU or LFU does not. When steady state indeed holds for some price function, Theorem 3.1 refomulated using the modified definitions of the costs applies. Note that the analogous interval caching schemes $\mathtt{LRU}_I^\pi$ and $\mathtt{LFU}_I^\pi$ are affected by the price function. For a particular value V they take different decisions than $\mathtt{LRU}_I$ and $\mathtt{LFU}_I$, respectively.

The next lemma shows that for a particular sequence s we can always find prices such that LRU or LFU is in ϵ-steady state when processing s. The experimental study in Section 8 shows that if we calculate prices based on one such s then it is likely to be good for other sequences s that span similar time periods.

Lemma 7.1 *Let s be a non-redundant sequence for* LRU (LFU). *There exists a price function $\pi_s(t) \ge 0$ such that* LRU *(resp.,* LFU*) with cache of size k is in ϵ-steady state (with threshold $C(s,k)$ close to 1) on s, where $\epsilon = 1/k$.*

Proof. We prove the lemma for LRU the proof for LFU is similar. We construct a price function such that when a page u is evicted (by LRU) at time t then $f_{\mathtt{LRU}^\pi}(u,t) = 1$. By the non-redundancy of the request sequence for all cached pages except the evicted one $f_{\mathtt{LRU}^\pi}(u,t) < 1$ at any time t. The π we define may be nonzero only until the time of the last eviction while serving s. We define π by induction on the number of evictions occuring while LRU serves the sequence. When there is only one eviction at time t' for a page u that had been cached when serving a request r, we define $\pi(t) = 1/(t' - t(r))$ for $t(r) \le t \le t'$, and $\pi(t) = 0$ for $t < t(r)$. Assume that there are k evictions while serving s. Let s' be the prefix of s up to the $(k-1)$-th eviction. Let t' be the time of the k-th eviction and t'' the time of the $(k-1)$-th eviction. We define $\pi(t)$ up to time t'' using the induction hypothesis on the prefix of s containing all the requests in s up to time t''. Let u be the page evicted

[12]As for LFU with this eviction function we evict the page with minimum eviction function value. For this algorithm to fit into the maximization framework we have to consider the negated eviction function.

at time t'. The proof now splits into one of the following two cases.
Case 1. Page u was last requested before time t'', i.e. $t(r) \leq t''$. Let $x = f_{\texttt{LRU}^\pi}(u, t'')$ We define $\pi(t) = (1-x)/(t' - t'')$ for every $t'' \leq t \leq t'$.
Case 2. $t(r) > t''$. We define $\pi(t) = 1/(t' - t(r))$ for every $t(r) \leq t \leq t'$ and $\pi(t) = 0$ for $t'' \leq t \leq t(r)$.
■

7.2 Incorporating prices into OPT′, GOPT and MPG

The results of Sections 4, 5, and 6 extend in a straightforward way to the setup when time is weighted by a price function. Here prices affect the decisions of both the replacement policies GOPT and MPG and their analogous interval caching schemes $\texttt{GOPT}_I$ and $\texttt{MPG}_I$.

The interval-caching algorithm $\texttt{GOPT}_I^\pi$ is defined analogously to $\texttt{GOPT}_I$ in Section 5.1 when we change the definition of the cost to incorporate prices. The cost of caching $u(r)$ from time $t(r)$ to the time $t(r) + \delta(r)$ of its next request is now

$$\mathrm{cost}(r) = s(r) \int_{t(r)}^{t(r)+\delta(r)} \pi(t) dt \ .$$

In the IRD caching model, the expected caching cost of a page $u(r)$ for time z depends on the distribution function $N_r(\delta)$ and on the price function

$$\mathrm{cost}_r(z) = \int_0^z \pi(x)(1 - N_r(x)) dx \ .$$

The interval-caching algorithm $\texttt{MPG}_I^\pi$ considers the expected cost as above, but otherwise, the definition of the utility function and the partition into intervals are as described in Section 6.2.

The proofs of Theorems 5.1 and 6.1 establishing convex optimality of $\texttt{GOPT}_I$ and $\texttt{MPG}_I$ extend in a straightforward way to $\texttt{GOPT}_I^\pi$ and $\texttt{MPG}_I^\pi$. Theorem 5.2 and 6.2 relating the performance of the interval-caching policies and the analogous replacement policies, and establishing that GOPT and MPG are nearly optimal when in ϵ-steady state, also extend in a straightforward way to $\texttt{GOPT}^\pi$, and $\texttt{MPG}^\pi$ respectively. These generalizations can be paraphrased as follows. If the prices $\pi(t) \geq 0$ are such that $\texttt{GOPT}^\pi$ is in ϵ-steady state (resp., $\texttt{MPG}^\pi$ is in δ, ϵ-steady state), then $\texttt{GOPT}^\pi$ (resp., $\texttt{MPG}^\pi$) are nearly optimal.

8 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 LRU (Least Recently Used), LFU (Least Frequently Used), the optimal policy OPT (Belady's algorithm), and the algorithm OPT′ defined in Section 4. In our experiments we assumed that all pages have the same sizes and fetching costs.

Our experiments were performed on a proxy trace recorded from AT&T research web proxy, which handled about 60,000 requests per day. We simulated LRU, LFU, OPT, and OPT′ using different cache sizes between 1000 and 5000 pages. Figure 2 shows the number of hits as a function of the cache size.

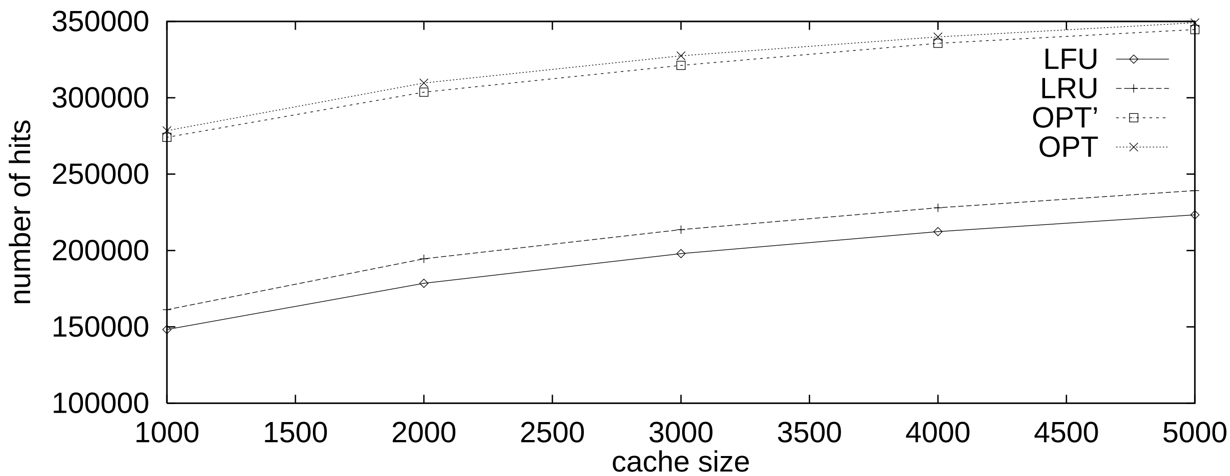


Figure 2: Number of hits as a function of cache size.

8.1 LRU

We measured the load on the cache for LRU whose eviction function $f_{\texttt{LRU}}(u,t)$, is defined as the time difference between t and the last request for page u before t. We also measured the load on the cache for the algorithm LRU′ that is equivalent to LRU but uses the eviction function whose value for a page u at time t, $f_{\texttt{LRU}'}(u,t)$, is the number of requests arriving between the time u was last requested before time t and time t. Our results show that the load on the cache for both algorithms is predictable. Therefore one can calculate prices that make both algorithms be in steady state most of the time. Our results also show that the load of LRU′ is better predictable than the load of LRU. Therefore one can calculate prices for this variant that facilitate steady state for a smaller ϵ.

To present our results we need the following notation. Let f_A be the eviction function of an algorithm A. Sort the cached pages at time t by increasing $f_A(u,t)$ values. Let $u_{\max}(t)$ denote the page with largest f_A value in the sorted order, Let $u_{0.8}(t)$ be the page at position $\lfloor 0.8k \rfloor$ in the sorted order, and let $u_{0.6}(t)$ be the page in position $\lfloor 0.6k \rfloor$ in the sorted order. Figure 3 shows three functions of time, $f_{\texttt{LRU}}(u_{\max}(t),t)$, $f_{\texttt{LRU}}(u_{0.8}(t),t)$, and $f_{\texttt{LRU}}(u_{0.6}(t),t)$. Figure 3(a) shows a daily activity cycle, where pages are cached for shorter time periods during normal working hours and for longer time periods during nights and weekends. Figure 3(b) zooms on one working day. For this working day we see that typically the oldest page in the cache was cached for about 600 seconds.

An even more stable behavior is demonstrated for LRU′. Figure 4 plots $f_{\texttt{LRU}'}(u_{\max}(t),t)$, $f_{\texttt{LRU}'}(u_{0.8}(t),t)$, and $f_{\texttt{LRU}'}(u_{0.6}(t),t)$ for a 12 day period. We observe that evicted pages during busy hours have eviction function value of about 6500 requests. At nights the value of the eviction function of evicted pages goes up to about 8000 requests and during the weekend it goes up to 10000 requests. Pages with eviction function value of 6000 or less are rarely evicted and fill almost at all times (except weekends) 80% of the cache. In our terminology we can say that even without a price function LRU′ almost at all times is in steady state with ϵ of 1/5 and threshold of 6000 requests. We expect that this value of ϵ can be reduced if we consider $\texttt{LRU}^{\pi}$ for an appropriately chosen price function.

a)

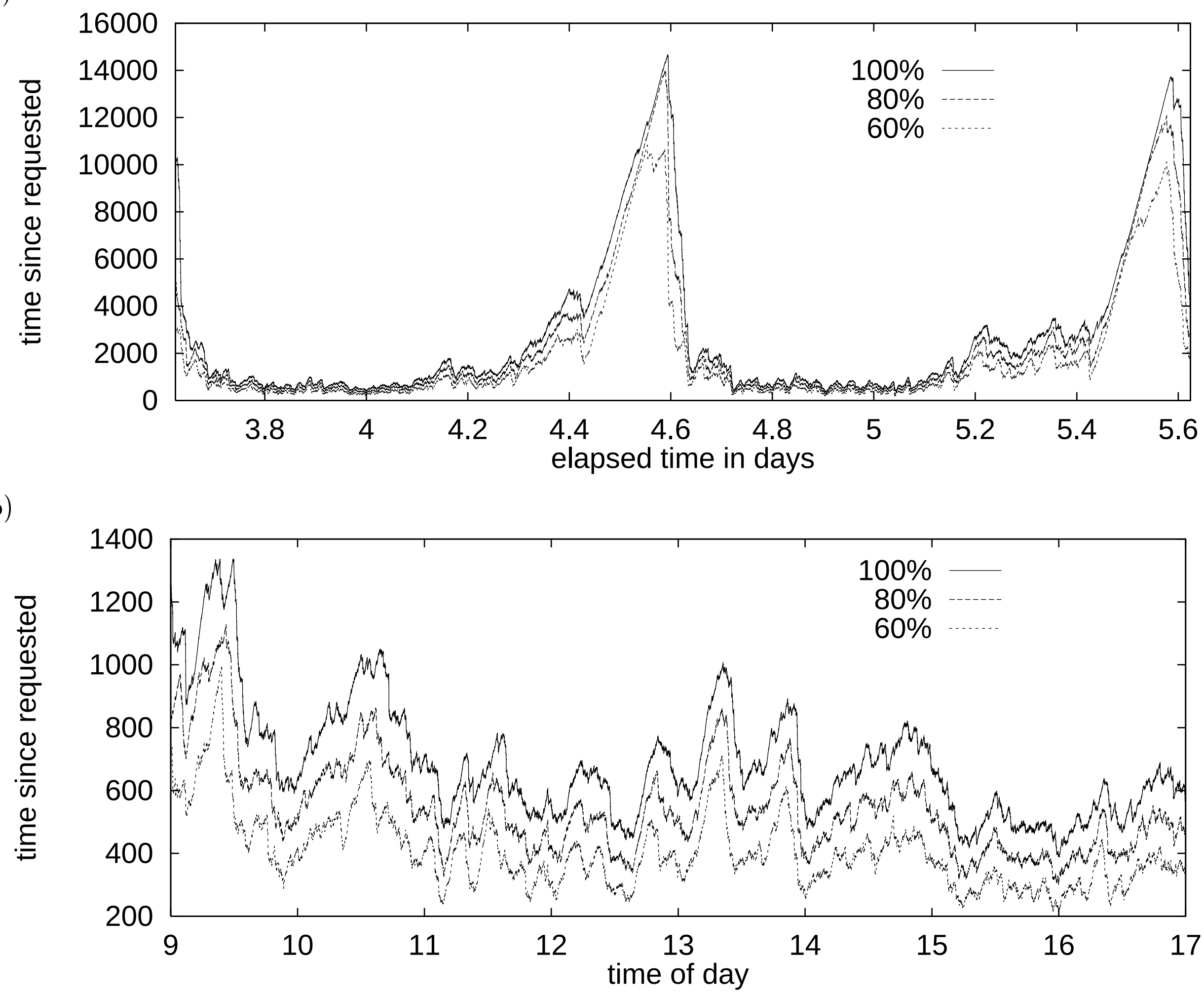


Figure 3: (a) The time in seconds that pages spend in cache during simulation of LRU for 2 working days on a cache of size 5000. The line marked 100% is $f_{\mathtt{LRU}}(u_{\max}(t), t)$. The other two lines are $f_{\mathtt{LRU}}(u_{0.8}(t), t)$, and $f_{\mathtt{LRU}}(u_{0.6}(t), t)$. (b) Enlargement of the previous graph for the period 9am-5pm of the first day.

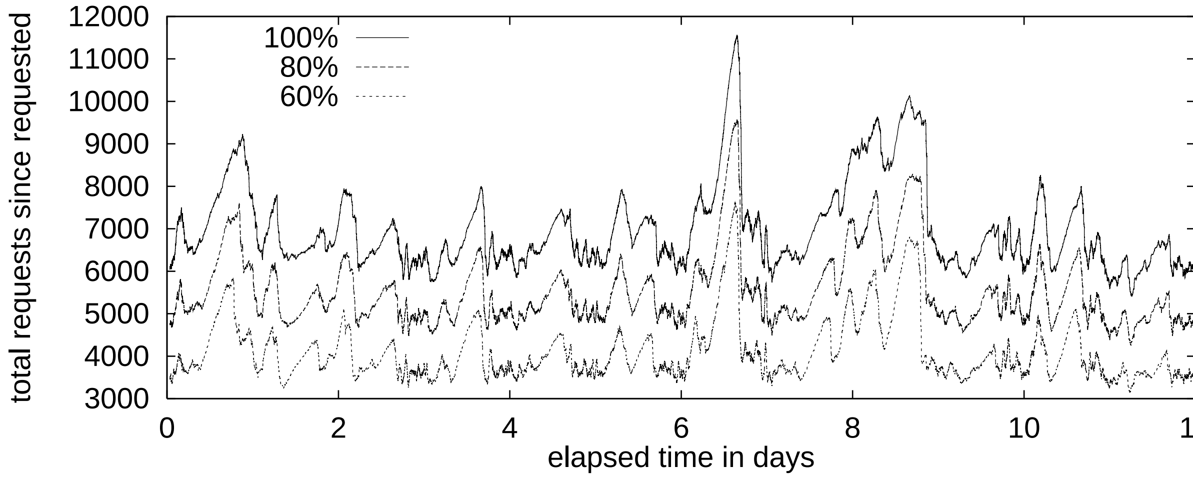


Figure 4: The number of requests that have arrived since a cached page was last requested for $\mathtt{LRU}'$ with cache size 5000. The line marked 100% is $f_{\mathtt{LRU}'}(u_{\max}(t), t)$. The other two lines are $f_{\mathtt{LRU}'}(u_{0.8}(t), t)$, and $f_{\mathtt{LRU}'}(u_{0.6}(t), t)$.

8.2 LFU

Recall the definition of the eviction function of LFU from section 2.1. We define $f_{\mathtt{LFU}}(u, t)$ to be the number of requests for page u before time t divided by t.[13] The algorithm LFU evicts the page which minimizes the eviction function. Therefore, here, we sort the pages in the cache at time t by *decreasing* $f_A(u, t)$ values. We define $u_{\min}(t)$ to be page in the cache with the smallest f_A value at time t, We also use $u_{0.8}(t)$ and $u_{0.6}(t)$ which are defined as in Section 8.1.

Figure 5 plots the functions $f_{\mathtt{LFU}}(u_{\min}(t), t)$, $f_{\mathtt{LFU}}(u_{0.8}(t), t)$, and $f_{\mathtt{LFU}}(u_{0.6}(t), t)$. The plot reflects 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 regular. Evicted pages during the first week always have an eviction function value below 1.3 at the time of eviction. Furthermore about 80% of the pages in the cache have eviction value above 1.3 during the first week. Therefore LFU is in steady state during the first week with ϵ of 1/5 and threshold 1.3.

8.3 OPT (Belady's algorithm) and OPT′

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 time is the furthest in the future. In other words, the value of the eviction function of the optimal algorithm, OPT, for a page u at time t, is the time

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

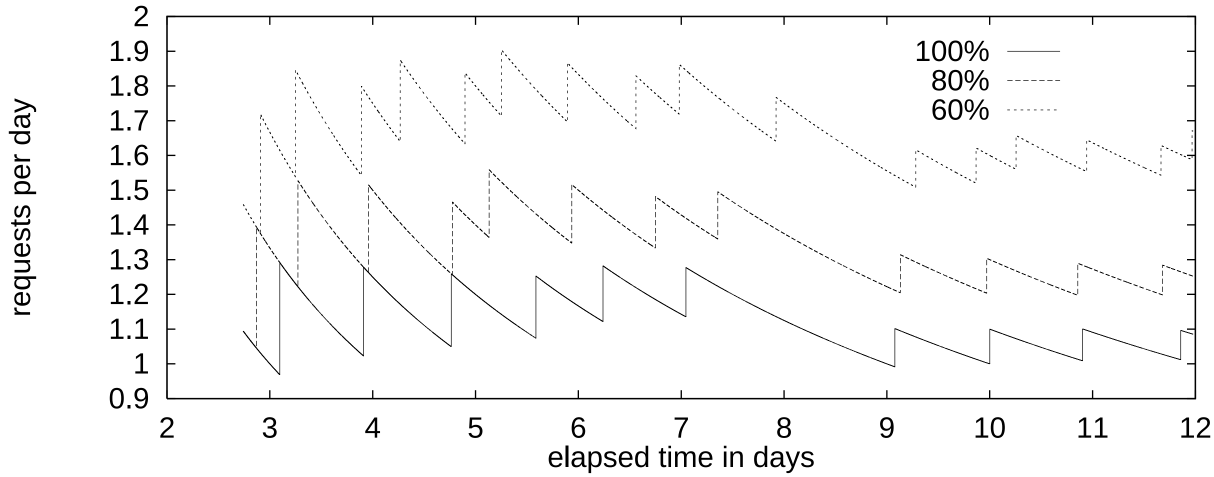


Figure 5: Request rate as a function of time for items in the cache while simulating LFU with cache of size 5000. The line marked 100% shows $f_{\texttt{LFU}}(u_{\min}(t), t)$, the line marked 80% shows $f_{\texttt{LFU}}(u_{0.8}(t), t)$, and the line marked 60% shows $f_{\texttt{LFU}}(u_{0.6}(t), t)$.

until the next request of u following time t. Figure 6 demonstrates a steady behavior of $f_{\texttt{OPT}}$ as a function of time. Here we show $f_{\texttt{OPT}}(u_{0.99}(t), t)$ instead of $f_{\texttt{OPT}}(u_{max}(t), t)$, where $u_{0.99}(t)$ is the page at position $\lfloor 0.99k \rfloor$ in the sorted order of the pages in the cache at time t. The curve $f_{\texttt{OPT}}(u_{0.99}(t), t)$ is smoother than the curve $f_{\texttt{OPT}}(u_{max}(t), t)$ that has a large number of long inter-request intervals. As in Sections 8.1 and 8.2 we also show $f_{\texttt{OPT}}(u_{0.8}(t), t)$, and $f_{\texttt{OPT}}(u_{0.6}(t), t)$. The plot shows that the functions $f_{\texttt{OPT}}(u_{0.99}(t), t)$, $f_{\texttt{OPT}}(u_{0.8}(t), t)$ and $f_{\texttt{OPT}}(u_{0.6}(t), t)$ are similar for different working days, establishing a predictable and even steady-state behavior.

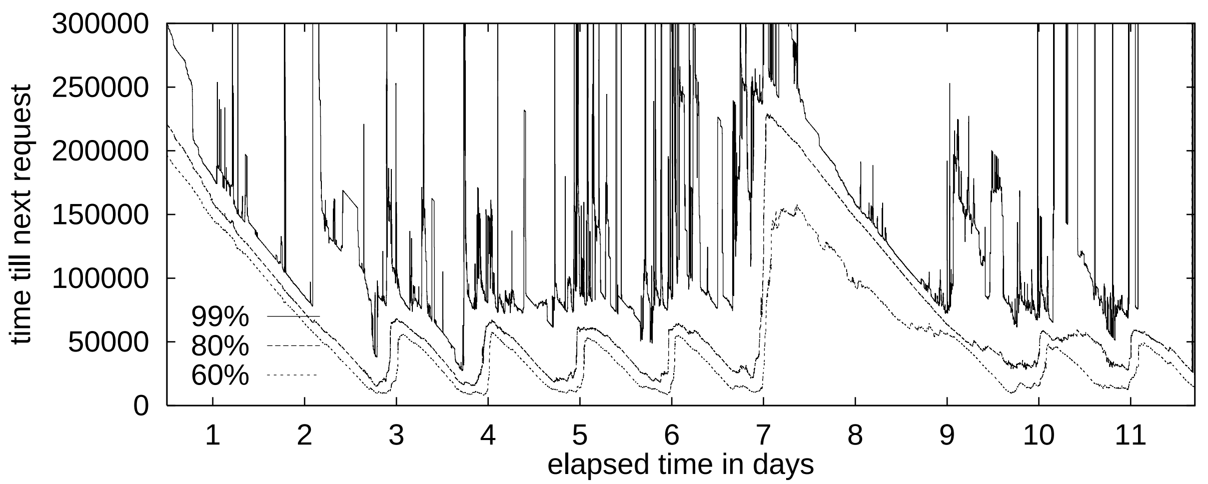


Figure 6: Time to the next request for items in the cache while simulating OPT with a cache of size 5000. The line marked 99% shows is $f_{\texttt{OPT}}(u_{0.99}(t), t)$, the line marked 80% shows $f_{\texttt{OPT}}(u_{0.8}(t), t)$, and the line marked 60% shows $f_{\texttt{OPT}}(u_{0.6}(t), t)$.

We proved in Section 4 that $\mathtt{OPT}$, $\mathtt{OPT}'$, and $\mathtt{OPT}_I$ have similar profit-cost tradeoffs when $\mathtt{OPT}'$ is in steady state. This result motivated us to check whether $\mathtt{OPT}'$ is in steady state on real request sequences, and how predictable is its load. Recall that the eviction function of $\mathtt{OPT}'$, $f_{\mathtt{OPT}'}(u,t)$, is the difference between the time of the next request for u following time t and the last request to u prior to time t. The value of this function is constant between consecutive requests for u. Figure 7 plots $f_{\mathtt{OPT}'}(u_{0.99}(t),t)$, $f_{\mathtt{OPT}'}(u_{0.8}(t),t)$, $f_{\mathtt{OPT}'}(u_{0.6}(t),t)$ where $u_{0.99}(t)$, $u_{0.8}(t)$, $u_{0.6}(t)$ are defined as before. The experiment shows that the load of $\mathtt{OPT}'$ is steady, even steadier than the load of $\mathtt{OPT}$. During weekdays $\mathtt{OPT}'$ is in steady state with $\epsilon = 1/5$, and threshold of 75000. During the weekend the load changes consistently in a predictable way.

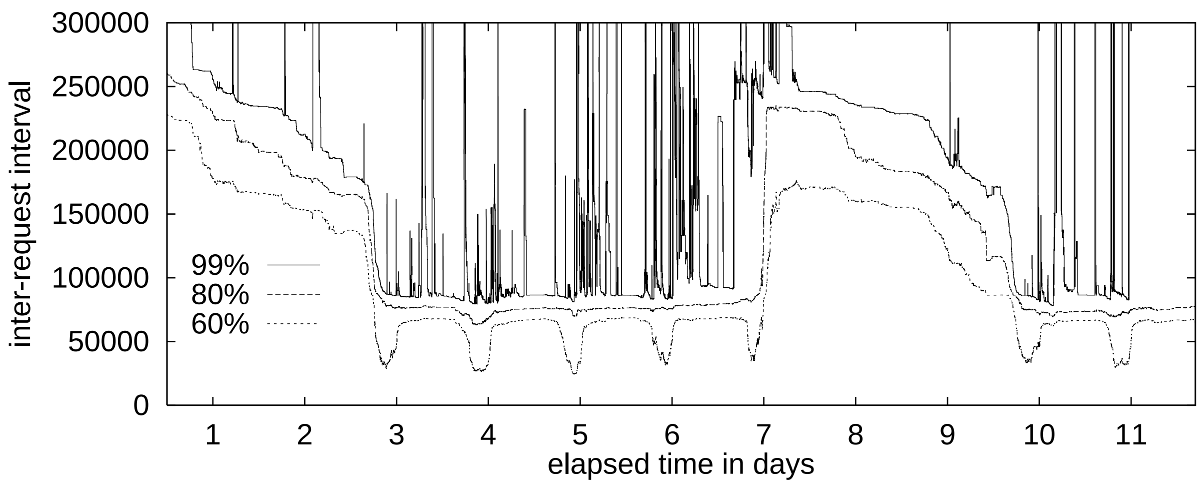


Figure 7: Interrequest intervals for items in the cache while simulating $\mathtt{OPT}'$ with a cache of size 5000. The line marked 100% shows $f_{\mathtt{OPT}'}(u_{0.99}(t),t)$, the line marked 80% shows $f_{\mathtt{OPT}'}(u_{0.8}(t),t)$, and the line marked 60% shows $f_{\mathtt{OPT}'}(u_{0.6}(t),t)$.

9 Discussion

Our experiments with LRU, LFU, OPT, and $\mathtt{OPT}'$ demonstrate regular and predictable behavior of their eviction function. Moreover, the algorithms are often in steady state for relatively long periods of time. Motivated by these experimental observations we fomalize the notion of a cache being in steady state, and prove performance guarantees for various algorithms under steady state conditions.

To establish our results we define the interval caching model and give interval caching algorithms that are analogous to the replacement policies which we consider. We prove that when the cache replacement algorithm is in steady state then its performance is similar to the performance of the corresponding interval caching algorithm. This correspondence between the regular model and the interval caching model under steady state conditions can be exploited to simulate cache replacement algorithms when only partial logs are available, as in [10]. Here we use this correspondance to prove performance guarantees for cache replacement algorithms under steady state conditions. In particular, we suggest a new cache replacement algorithm, that we call `MPG` (minimum profit gradient) for caching when the distribution of the next request time of the requested page is known. We show that `MPG` is nearly optimal when the load is predictable.

We extended our results to the case when the cache is not in steady state but the load on the cache changes in a predictable way. We suggest to model the load on the cache by a price function $\pi(t)$ such that after weighting time by $\pi(t)$ the algorithm in question is in steady state. The question of finding a good way to calculate these prices for a particular algorithm is left for further research.

Acknowledgments

We thank the anonymous referees for their perceptive suggestions.

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] P. Cao and S. 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. Exploiting regularities in Web traffic patterns for cache replacement. In *Proc. 31st Annual ACM Symposium on Theory of Computing*. ACM, 1999.

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

[10] 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.

[11] 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.

[12] 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.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman and Co., San Francisco, CA, 1979.

[14] 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`.

[15] 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.

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

[17] 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.

[18] 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`.

[19] 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`.

[20] 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`.

[21] 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`.

[22] 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.

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