Evaluating Server-Assisted Cache Replacement in the Web

Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford

AT&T Labs–Research, 180 Park Avenue, Florham Park, NJ 07932 USA,
E-mail: {edith,bala,jrex}@research.att.com
Web page: http://www.research.att.com/~{edith,bala,jrex}

Abstract. [1] To reduce user-perceived latency in retrieving documents on the world wide web, a commonly used technique is caching both at the client's browser and more gainfully (due to sharing) at a proxy. The effectiveness of Web caching hinges on the replacement policy that determines the relative value of caching different objects. An important component of such policy is to predict next-request times. We propose a caching policy utilizing per-resource statistics of inter-request times. Such statistics can be collected either locally or at the server, and piggybacked to the proxy. Using various Web server logs, we compared existing cache replacement policies with our *server-assisted* schemes. The experiments show that utilizing the server knowledge of access patterns can greatly improve the effectiveness of proxy caches. Our experimental evaluation uses an *interval caching* model. Interval caching values the utility of a unit of cache storage as a function of time. Instead of the usual tradeoffs of *profit* (combined value of cache hits) and cache size we measure tradeoffs of profit and *caching cost* (average cache size). Interval-caching policies have analogous cache replacement policies, and policies and their relative performance in the two models are closely tied. Interval-caching allowed us to evaluate and compare different replacement policies by using *server logs*, without having to construct a full workload model for each client's cache.

1 Introduction

The popularity of the World Wide Web has imposed a significant burden on the communication network and web servers, leading to noticeable increases in user-perceived latency. Caching popular resources at proxies offers an effective way to reduce overhead and improve performance. Given that proxy caches are of a finite size, sound policies are needed for replacing resources in the cache. Although cache replacement has been studied extensively in the context of processor architecture and file systems, the Web introduces more complicated challenges, since resources vary substantially in their sizes, fetching costs, and access patterns. Indeed, new replacement policies have been developed to incorporate these parameters [1, 2, 3, 4]. Although the Web exhibits high variability between the

[1] Revision of an ESA '98 paper

access patterns of different resources, individual resources have more regular acccess patterns. The most important factor for a good replacement policy remains predicting the next request time of a resource which depends on the ability to observe and interpret the access patterns.

The natural place to collect statistical information on request patterns and interdependencies is the Web server. Any one proxy views a small number of requests to any one server, and hence, the data at its disposal may not be sufficient to model the access patterns. In addition, the proxy contacts a large number of different servers, and the required computational efforts to generate such models may not be justified. Proxy cache performance can be enhanced by having servers generate predictions of accesses to their resources and communicate this knowledge back to the proxies (as hinted in [5] and explored in [6]). Following the approach in [6, 7, 8], we assume that proxies and servers can piggyback information on regular request and response messages. The information exchange can be incorporated into the HTTP/1.1 protocol without disrupting the operation on non-participating proxies and servers [6]. For the sake of efficiency, the server does not maintain an online history of each proxy's past accesses and, hence, bases its response to the proxy only on the requested resource and piggyback content. The protocol does not involve any new messages between the proxy and server sites. Operating within these practical constraints, we propose policies for *server-assisted* cache replacement.

Performance of cache replacement policies hinges on the accuracy of the predictions of the next access time for each resource. The `LRU` policy, for example, utilizes the last request time of each resource as the only predictor for its next request. The `LFU` replacement policy utilizes the frequency with which a resource is accessed and the `LRV` policy [4] utilizes the number of previous requests to the resource. We propose to enhance replacement policies by providing the proxy with the distribution function of the next request time for each resource. The distribution is estimated by collecting per-resource statistics on inter-request times. In server-assisted replacement, the server generates a histogram of inter-request times by observing its request logs. The histogram is piggybacked to the requesting proxy and used to determine the duration for caching resources. In addition, the server may piggyback hints about related resources that are likely to be accessed in the future.

Our experimental evaluation compares server-assisted and *proxy-local* (i.e., without the help of server hints) replacement policies. The evaluation of server-assisted policies necessitated the use of *server logs* since proxy logs do not provide us the statistics available to the servers. We used three large Web server logs. The logs provide, for each client, the full request sequence of resources belonging to the particular server. We used *interval caching model* to evaluate the performance of different replacement policies using the information in the server log, without directly considering (or simulating) the full workload of the cache. The *price* is a function of time capturing the current value of a unit of cache storage. The *caching-cost* of a resource for a time period is the product of the resource size, the length of the time interval, and the average price during that period. In the experimental evaluation we used constant-valued price functions (i.e., an assumption of *steady-state*). Instead of considering the usual tradeoffs of cache size and *profit* (sum of the fetching cost of objects across all cache hits), we measured tradeoffs of profit and *average cache portion allocated to server's resources.*[2] The profit and caching costs are computed for each client separately and summed over all clients. Tradeoffs are obtained by varying the cache price. In the interval caching model, cache replacement policies are viewed as generating a sequence of decisions on if and how long to cache each resource. Traditional cache replacement policies (such as `LRU`) have analogous interval caching policies [9]. We use and compare the analogous policies in the evaluation.

Related Work: Replacement policies were previously analyzed under some restriction or statistical assumptions on the data [10, 11]. Adaptive policies that capture and use reference locality for replacement decisions were studied in [12]. Adaptive policies utilizing distributions combined with an interval-caching model were previously used in various contexts. Lund et al. [13] and Keshav et al.[14] proposed policies for virtual circuit holding-time for IP over ATM virtual circuits, based on statistics of the packet inter-arrival sequence. Other contexts were spinning on a lock in a shared memory multiprocessor [15], adaptive disk spindown [16], and subsequent to our work, persistent-connection timeouts [17]. Theoretical analysis was provided in [9, 15, 16, 13]; an optimal interval caching policy was given in [15, 14, 9]; and the analogous optimal policy under a fixed-cache size, for predictable load conditions, was provided in [9]. The optimal policy in this context uses knowledge of the distributions in the best possible way.

In Section 2 we formulate the optimal caching policy in the interval caching model where each resource has a distribution of inter-request times. We discuss a replacement policy when knowledge of this distribution is replaced by a sample of inter-request times. Section 3 provides an experimental evaluation of different policies. We conclude with a section on ongoing and future work.

2 Cache Replacement Model

The interval caching framework models the value of caching a resource in terms of its fetching cost, size, next request time, and *cache prices* during the time period between requests. It is possible to set cache prices according to the *load* such that the interval caching algorithm emulates the analogous traditional cache replacement algorithm with bounded cache size[9]. After deriving expressions for the profit and the cost of caching a resource, we describe how the Web server can aid the proxy by estimating the request interarrival distribution and the likelihood of accesses to related resources. Together, the proxy's price function and the inter-request statistics guide the cache replacement decision.

[2] Total caching cost under steady-state, divided by elapsed time.

2.1 Cache Price Function and Resource Utility

As the basis of the cache replacement policy, we use the *price function* $\pi(t)$ of the cache and the *utility* of a resource to the proxy. For a resource r requested at time t, let s_r be the size, $t + \Delta_{r,t}$ be the time of its next request, and $f_{r,t}$ be its fetching cost at time $t + \Delta_{r,t}$. If $\Delta_{r,t}$ is known in advance, then the resource is worth caching for either 0 or $T = \Delta_{r,t}$ seconds. Intuitively, the proxy should favor the caching of resources with high fetching cost and/or small size. We define the *utility* of caching the resource for T seconds as

$$u_{r,t}(T) = \frac{f_{r,t}}{s_r \int_t^{t+T} \pi(t)dt} .$$

When caching decisions are made with respect to a price-function and a threshold value, the resource is cached for a time period if its utility exceeds the threshold throughout the interval.

In reality, the proxy has only partial or statistical knowledge of fetching costs and next request time of cached objects. When making the caching decisions, we use *estimated utility* for caching a resource for a particular time period. The fetching cost $f_{r,t}$ can be based on the last fetching time, the network load on the path to the server, or the server load. The most challenging part of estimating resource utility is to obtain a good estimate of the next request time. The various cache replacement policies make implicit assumptions about this value, and their performance hinges on the quality of (absolute or relative) estimates for $\Delta_{r,t}$. For example, the `LRU` replacement policy assumes $f_{r,t}/s_r$ to be identical and fixed for all resources and utilizes an estimate on $\Delta_{r,t}$ that is decreasing with the last request time for r. The Greedy-Dual-Size algorithm [1] (generalization of `LRU` that incorporates fetching cost and sizes) can be viewed in the above framework as assuming a steady-state and substituting $t - t_r$ for $\Delta_{r,t}$, where t_r is the last request time of r.

Instead of assuming a particular expression for the next request time, we formulate an optimal request sequence when the time between requests is captured by a probability density function $\Delta_{r,t}(x)$ $(x \geq 0)$ (probability density that the object is requested in time $t + x$) and let $L_{r,t}(x)$ be the corresponding distribution function. Let $f_r(x)$ be the expected fetching cost of r at time $t + x$. We compute the expected profit and cost of caching the resource for T time units (or up till its next request). The estimated utility of caching the object for T time units is the ratio of the expected profit and expected cost. The expected profit is

$$\text{profit}_{r,t}(T) = \int_0^T \Delta_{r,t}(x) f_r(x) dx$$

and the expected cost is

$$\text{cost}_{r,t}(T) = s_r \int_0^T \pi(t + x)(1 - L_{r,t}(x))dx .$$

The estimated utility of increasing caching time from T_1 to T_2 is:

$$\frac{\text{profit}_{r,t}(T_2) - \text{profit}_{r,t}(T_1)}{\text{cost}_{r,t}(T_2) - \text{cost}_{r,t}(T_1)} = \frac{\int_{T_1}^{T_2} \Delta_{r,t}(x) f_r(x) dx}{s_r \int_{T_1}^{T_2} \pi(t+x)(1 - L_{r,t}(x)) dx} \ .$$

If V is the threshold then the caching time T of a resource can be predetermined as follows. Caching for T' time is *valid* if the estimated utility of increasing caching period from y to T' exceeds V for all $y \in [0, T']$.

$$\forall y < T', \frac{\text{profit}_{r,t}(T') - \text{profit}_{r,t}(y)}{\text{cost}_{r,t}(T') - \text{cost}_{r,t}(y)} \geq V \ .$$

Among all valid T', we select T to maximize the expected profit, that is, we select the maximum valid T'. Formally,

$$T = \arg\max_{T'} \quad T' \text{ is valid} \tag{1}$$

The value of T serves as a expiration timeout for evicting resource r from the proxy cache.

The optimal interval-caching policy maximizes cache hits for a given caching cost. When distributions are used, the optimal policy maximizes expected number of cache hits for a given expected cost. The optimal policy has the best performance among all policies with full knowledge of the underlying probability space from which sequences are drawn, but without prior knowledge of the actual sequence. Cohen and Kaplan [9] show tight correspondence between these tradeoffs and optimal tradeoffs of hits and cache size obtained for the analogous policies. In particular, under reasonable assumptions it is possible to choose cache prices such that the interval-caching algorithm emulates a limited cache size.

2.2 Utilizing Inter-Request Statistics

By receiving requests from a large number of clients, servers can estimate the distribution $\Delta_{r,t}(x)$ of the time between successive requests. In responding to a request for resource r, the server can piggyback the probability distribution (in a discretized form) to aid the proxy in its cache replacement decisions. The distribution is computed using a portion of the server logs. Consider discrete times $0 = t_0, t_1, \ldots, t_N$ and a proxy cache in steady-state ($\pi(t)$ constant) with fetching costs that do not change across time ($f_r(x) \equiv f_r$). For a resource r let s_r be its size and f_r its fetching cost. Let n_r be the total number of requests for r under consideration in the server logs. Let $c_r(i)$ be the number of requests for r such that the previous request for r by same client preceded by t_i to t_{i-1} seconds. Let P_i be the price per unit-size of caching for t_i time. Then the estimated expected cost of caching for time-length t_i is approximately

$$\text{cost}_r(i) = \frac{s_r}{n_r}(P_i \sum_{j \geq i} c_r(j) + \sum_{j < i} c_r(j) P_j)$$

(in the experiments we compute the exact value, where the second term is replaced by the sum of prices per unit-size over all inter-request times smaller than t_i) and the estimated expected profit is

$$\text{profit}_r(i) = \frac{f_r}{n_r} \sum_{j \le i} c_r(j) \ .$$

To compute the recommended time interval i we start with $i = 0$ and repeat: we look for the minimum $j > i$ such that

$$\frac{\text{profit}_r(j) - \text{profit}_r(i)}{\text{cost}_r(j) - \text{cost}_r(i)} \ge V$$

and set $i \leftarrow j$. If no such j exists, we stop.

Contents that corresponds to a particular URL change over time. We propose that inter-request statistics are collected per-URL but that the expected next-modification time of the contents is incorporated in the distribution function (by nullifying the density function for values exceeding the modification time and moving the "mass" to infinity).

The use of server-generated predictions implicitly assumes that the set of clients have similar access patterns. The server can increase the accuracy of the predictions by classifying various "client types" based on parameters such as their volume of traffic to the server and time-of-day at the proxy and the server. Statistics such as access frequency of resources can be collected for each class separately. The proxy identifies its "type" to the server (e.g., provides it with estimate on number of daily/weekly requests made to server resources) and server returns statistical information about the next request time, along with the request resource. The more accurately the server-generated distribution function fits the access patterns of a particular client/resource, and the less variance these distributions exhibit, the better are the replacement sequences generated. However, dividing the proxies into too many types also decreases the amount of data available for estimating distributions, which results in lower accuracy and higher computational complexity. (The extreme partition is assigning a type for each client. This is essentially a local policy that does not utilize the server's additional knowledge.)

The above formulation associates with each resource a distribution on its next request time. In reality, different resources are dependent, and conditional probabilities capturing these interdependencies may yield more accurate predictions. In [6], we proposed heuristics for constructing *volumes* that capture the pairwise dependencies between resources by observing the stream of requests at a server; measuring such dependencies was also suggested by [18, 19] and further developed in [6]. Let $p_{s|r}$ be the proportion of requests for resource r that are followed by a request for resource s by the same client within T seconds. Resource s is included in r's volume if $p_{s|r}$ is greater than or equal to a threshold probability p_t. When a proxy requests resource r, the server constructs a piggyback message from the set of resources s with $p_{s|r} \ge p_t$. Based on these predictions, the proxy can extend the expiration time (or adjust the priority) of resources that appear in the piggyback message and are stored in its cache. After estimating the implication probabilities $p_{s|r}$, the server can evaluate the potential effectiveness of piggyback information by measuring how often a piggyback message generates a *new* prediction for resource s (particularly important when an access to s is often preceded by a *sequence* of requests by the same proxy).

3 Performance Evaluation

To compare the *proxy-local* and *server-assisted* cache replacement policies, we evaluated the various schemes on three large server logs. The server logs can be viewed as providing a sequence of triples, with the requesting client, requested resource, and request time. For this initial study, we assumed constant price functions and that all resources have the same size and fetching cost; hence, the profit is simply the number of cache hits and the cost is the total time resources spend in cache. We evaluated the cache hit ratio versus the mean cost per hit, averaged across all requests. These measures are computed by partitioning the server logs by clients, and computing the number of hits and the total time-in-cache for each resource and for each client. The latter quantities are then summed over clients and resources. The total hit ratio is obtained by dividing the total number of cache hits by the total number of requests. The cost per hit is obtained by dividing the total cost (object-seconds) by the number of hits.

3.1 Server Logs

The experiments use the access logs of three Web servers, from Amnesty International USA, Apache Group, and Sun Microsystems. The server logs represent a range of Web sites in terms of the number of resources and accesses, as shown in Table 3.1. Though the servers do not necessarily see the requests that were satisfied directly at the client or proxy caches, the logs do include all `If-Modified-Since` requests to validated cached copies of resources. Many clients have very short interactions with a server, resulting in a small number of requests. These clients experience very low cache hit rates, even under an optimal replacement policy. In the AIUSA log, only 11% of requests were for resources already requested by same client. The corresponding figures are 36% and 38% for the Apache and Sun logs, respectively. These numbers provide an upper bound on the cache hit ratio for all of the cache replacement policies.

3.2 Cache Replacement Policies

We examined a variety of replacement policies, including an optimal omniscient policy, three proposed proxy-local policies, and our server-assisted policies. Each policy is described in terms of our framework, assuming constant price functions:

Optimal (`OPT`*):* The optimal replacement sequence caches all resources with a next request time within some fixed threshold value. Cost-performance tradeoffs are obtained across a range of threshold values. `OPT` provides a good yardstick for gauging other policies.

Log (days)	Number of Requests	Number of Clients	Requests per Source	Unique Resources Considered
AIUSA (28)	180,324	7,627	23.64	1,102
Apache (49)	2,916,549	271,687	10.73	788
Sun (9)	13,037,895	218,518	59.66	29,436

Table 1. Server log characteristics

Fixed (`LRU`): `LRU` predicts a resource's next request time based only on the last request time. This results in a proxy-local policy that keeps all resources in the cache for the same, fixed, period of time. Cost-performance tradeoffs are measured by varying the resource expiration times across a range of values from 40 seconds to 24 hours.

Exponentially averaged mean and variance (`EMV`): This proxy-local policy is based on the premise that current inter-request time interval of a resource is more correlated with recent inter-request times. This policy was evaluated by Keshav et al. [14] on circuit holding times. For a parameter $0 < \alpha < 1$ (we used $\alpha = 0.3$), the policy maintains exponentially averaged estimates on the mean and variance of the inter-request times: initially $\mu_0 = 0$ and $\sigma_0 = 0$. After observing the kth inter-request time, t_k, we compute

$$\mu_{k+1} = \alpha t_k + (1-\alpha)\mu_k$$
$$\sigma_{k+1} = \alpha|\mu_k - t_k| + (1-\alpha)\sigma_k$$

A resource is cached for $\mu_k + 2\sigma_k$ time units, when this is below the threshold.

Local inter-request statistics (`LIS`): This is the policy outlined in Section 2.2, where the inter-request times histogram is constructed using locally-available (at the client) data. If fetching costs and resource sizes are uniform, this local policy reduces to the *adaptive policy* proposed in [13, 14] for circuit holding times. Typically, resources are requested only a few times by any one client in the duration of the log. When predicting an inter-request interval for resources requested 15 or fewer times, we used a histogram containing all *other* inter-request time intervals. Otherwise, the histogram contained all inter-request times. Future inter-request times were included in order to account for the initial lack of history. Note that `LIS` does not cache resources requested two or fewer times. The histogram bin partition used to evaluate `LIS`, `SIS` and `SIS-c` policies (see below) consisted of 21 time intervals varying from 20 seconds to 24 hours.

Server inter-request statistics (`SIS`): The server-assisted policy outlined in Section 2.2. The server generates inter-request distributions for each resource r using statistics from all its clients. For each resource and threshold value U, there is a "recommended caching period" $t_r(U)$. Tradeoffs are obtained by sweeping U.

Server inter-request statistics, filtered by client type (`SIS-c`*):* This policy is a refinement of `SIS`, where the server generates separate inter-request distributions and caching periods for different types of clients, based on how often they issue requests to the server. The statistics obtained for each type are applied to clients of that type. The experiments consider three classes for the AIUSA logs, four for the Apache logs, and five for the Sun logs.

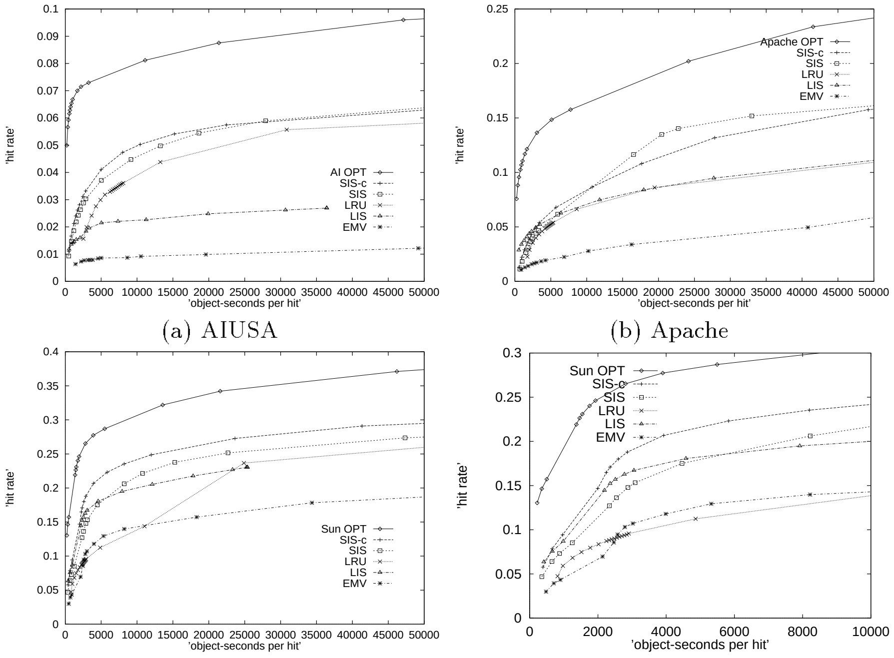
Volume enhanced (VOL): This enhancement captures interdependencies between server resources, and can be combined with any of the other cache replacement policies. We evaluate the `LRU+VOL` policy, which is the `LRU` policy (each object is kept for the same amount of time) with the following enhancement: When a resource r appears in a piggybacked volume, and is currently cached, its expiration time is extended by its stated caching time. We also evaluate `SIS+VOL`, a similar enhancement of the `SIS` policy.

3.3 Cost-Performance Tradeoffs

The graphs in Figure 1 compare the `OPT`, `LRU`, `EMV`, `LIS`, `SIS`, and `SIS-c` algorithms on all three server logs, across a range of thresholds. Each threshold value results in a single measure of performance (the hit rate on the y-axis) and overhead (object-seconds-per-hit on the x-axis). The cost per hit increases with the hit ratio for all of the replacement policies, since small improvements in the hit ratio incur progressively larger costs. As a result, even the `OPT` algorithm incurs a significant cost per hit as the hit ratio grows closer to the upper bound (11%, 36%, and 38%, respectively, for the AIUSA, Apache, and Sun logs). The graphs also show that the server-assisted `SIS` and `SIS-c` policies typically outperform the three proxy-local policies (`LRU`, `EMV`, and `LIS`). Consequently, server-assisted policies achieve a higher hit ratio for the same cache size, or the same hit ratio for a lower cache size, compared to proxy-local schemes. Even `SIS`, where the server supplies the same hints per-resource for all clients, outperforms the `LRU` scheme that treats all resources uniformly.

Among local policies, `LRU` outperforms the more involved `LIS` and `EMV` when the cost-per-hit is high. This seemingly counter-intuitive behavior is due to the large number of resource-client pairs with small number of requests. `LIS` does not cache a resource requested by a client twice or less, and `EMV` does not cache a resource on its first request. Hence, `LIS` and `EMV` are subject to a lower performance ceiling than `LRU`. Furthermore, when there are only a few requests, the selection of a caching period is based on very limited statistics and is less accurate. The above suggests that a local policy which combines `LIS` or `EMV` with a default non-zero caching period would enhance its performance.

The policies `SIS`, `SIS-c`, and `LIS` differ in the partition of the clients used to collect the inter-request intervals statistics. The granularity of the client partition corresponds to a tradeoff between the amount of statistics available and the applicability of the statistics to a particular client. The two extremes are `LIS`, which collects statistics locally at each client, and `SIS` which collects the same



(a) AIUSA (b) Apache

(c) Sun (c1) Sun (zoomed in at low-cost hits)

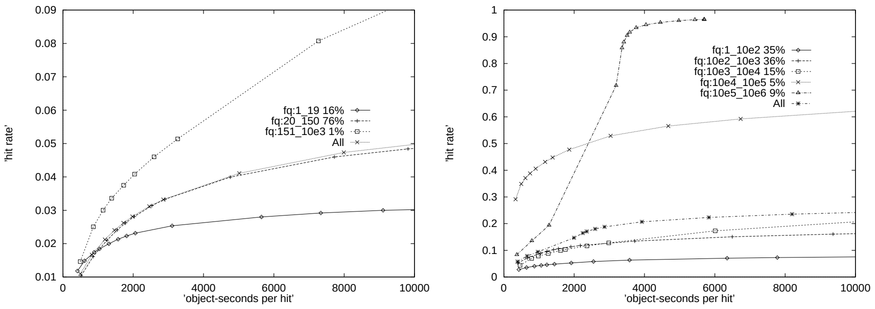
Fig. 1. Hit rate vs. cost-per-hit for all six replacement policies

statistics for all clients. `SIS-c` differentiates between different types of clients, based on total number of requests.

On very low ranges of object-seconds-per hit (see the zoomed in portion of Figure 1), the local policy `LIS` outperforms other policies. This is more pronounced for the Apache and in particular the Sun logs, where client-resource interactions are on average longer (and `LIS` indeed exhibits better relative performance). The explanation is that resources that have smaller inter-request times, and hence, lower expected cost-per-hit, tend to be requested many times by clients and hence, there is a good amount of inter-request statistics available locally. As the cost-per-hit increases, `SIS` and `SIS-c` significantly outperform `LIS`, indicating that locally-available statistics are not sufficient. `SIS-c` typically outperforms `SIS`, substantiating the value of differentiating between client types. The above results suggest a combined policy likely to dominate all three: For each client, each resource is considered separately; if there is "enough" local statistics, we use `LIS`; otherwise, the client utilizes the statistics of its client-type (`SIS-c`) or even the statistics of all clients (`SIS`). The downside of the combined policy (and of `LIS`) is that each entity in the hierarchy needs to collect inter-request statistics.

Figure 2 shows the performance of the `SIS-c` policy on various client classes. For example, the top curve in Figure 2(a) shows the cost-performance tradeoffs for the top clients which accessed the AIUSA server 151–1000 times during the

28-day period. These clients also exhibit much higher hit rates than the other clients. This is also true under other replacement policies, such as `OPT`, since these clients are much more likely to access the same resource multiple times. These accesses stem from high-end users or, perhaps, from proxy sites that relay requests for multiple clients. It is precisely these clients or proxies that would most benefit from participating in an enhanced information exchange with the server sites.



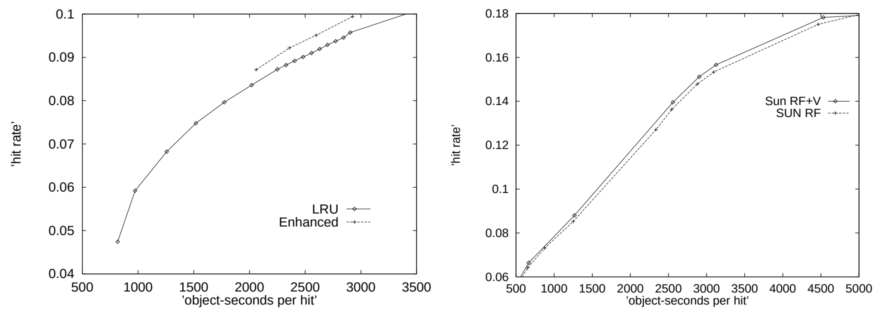
(a) AIUSA (b) Sun

Fig. 2. Hit rate vs. cost-per-hit for `SIS-c` client classes

In Figure 3, we investigate the benefits of using server-generated volumes to extend the expiration time of cached resources. The piggyback-enhanced replacement policies utilize volumes obtained with time interval $T = 300$, probability threshold $p_t = 0.25$, and effective probability 0.2 [6]. Incorporating the piggyback information improves the performance of the `LRU` policy, as shown in Figure 3(a). The volume information augments the proxy-local policy with information about resource access patterns. This extra knowledge is less useful for the `SIS` policy, as shown in Figure 3(b), since the basic `SIS` scheme already gleans useful information from the server estimates of the inter-request times for individual resources. In fact, the basic `SIS` scheme even outperforms the volume-enhanced `LRU` policy, suggesting that accurate estimates of inter-request times may be more useful to the proxy than predictions about future accesses to other resources. Further experiments with other server logs and volume parameters (e.g., larger p_t) should lend greater insight into the cost-performance tradeoffs of volume-enhanced cache replacement policies.

4 Summary and future work

Our experiments implicitly assume that the servers see all or most of the client requests. This becomes less likely as proxy caching becomes more common, particularly when the proxy employs cache coherency policies that avoid generating `If-Modified-Since` requests to servers. By hiding client accesses from the server, these trends potentially limit the server's ability to generate accurate



(a) LRU vs. LRU+VOL (b) RF vs. RF+VOL

Fig. 3. Volume-enhanced policies on the Sun log

predictions of inter-request distributions and dependencies between resources. One approach is for the server to collect statistics based on the fraction of traffic due to low-volume clients. These clients are more likely to be individual users with only browser caching. As an extension to our server-assisted cache replacement model, we are also considering ways for participating proxies to summarize information about client requests that are satisfied in the cache. The server can accumulate these additional statistics across multiple proxy sites to generate better estimates of the inter-request distributions and resource dependencies.

References

[1] 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`.

[2] 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.

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

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

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

[6] 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*, Sept. 1998.

[7] B. Krishnamurthy and C. E. Wills, "Study of piggyback cache validation for proxy caches in the world wide web," in *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, (Monterey, California), December 1997.
`http://www.usenix.org/events/usits97`.

[8] B. Krishnamurthy and C. E. Wills, "Piggyback server invalidation for proxy cache coherency," in *Proceedings of the World Wide Web-7 Conference*, April 1998.

[9] 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.

[10] 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.

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

[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] 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`.

[14] 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*, vol. 13, 1995.

[15] A. R. Karlin, K. Li, M. S. Manasse, and S. Owicki, "Empirical studies of competitive spinning for a shared-memory multiprocessor," in *Proceedings of the 13th ACM Symposium on Operating Systems Principles*, 1991.

[16] P. Krishnan, P. M. Long, and J. S. Vitter, "Adaptive disk spindown via optimal rent-to-buy in probabilistic environments," *Algorithmica*, vol. 23, pp. 31–56, 1999.

[17] E. Cohen, H. Kaplan, and J. D. Oldham, "Policies for managing TCP connections under persistent HTTP," in *Proceedings of the World Wide Web-8 Conference*, 1999.

[18] A. 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.

[19] V. N. Padmanabhan and J. C. Mogul, "Using predictive prefetching to improve world wide web latency," *Computer Communication Review*, vol. 26, no. 3, pp. 22–36, 1996.
`http://daedalus.cs.berkeley.edu/publications/ccr-july96.ps.gz`.