Variable QoS from Shared Web Caches: User-Centered Design and Value-Sensitive Replacement*

Terence P. Kelly Sugih Jamin
{tpkelly,jamin}@eecs.umich.edu
Electrical Engineering & Computer Science

Jeffrey K. MacKie-Mason
jmm@umich.edu
Economics & School of Information

University of Michigan
Ann Arbor, Michigan 48109 USA

November 12, 1999

Abstract

Due to differences in server capacity, external bandwidth, and client demand, some Web servers value cache hits more than others. Assuming that a shared cache knows the extent to which different servers value hits, it may employ a value-sensitive replacement policy in order to generate higher aggregate value for servers. We consider both the *prediction* and *value* aspects of this problem and introduce a novel value-sensitive LFU/LRU hybrid that biases the allocation of cache space toward documents whose origin servers value caching most highly. We compare our algorithm with others from the Web caching literature and discuss from an economic standpoint the problems associated with obtaining servers' private valuation information.

1 Introduction

Since the inception of the World Wide Web, caching has helped to reduce server load, network traffic, and latency at the client end. However, among researchers no clear consensus exists on the relative importance of these goals or of related performance metrics such as hit rate and byte hit rate. Furthermore, Web caches currently provide only "best-effort" service, in the sense that they do not account for possible differences in the extent to which system stakeholders (clients, servers, and ISPs) value caching.

We begin with the premise that Web cache performance is best measured in terms of user satisfaction, and we conjecture that system users are heterogeneous with respect to the value they receive when their documents are served from cache. Finally, we observe that storage space in *shared* Web caches—proxies serving corporate- or campus-sized LANs and backbone caches embedded in high-speed networks, as opposed to browser caches—can be diverted to serve those who value caching most by removal policies sensitive to heterogeneous user preferences. Caches are ideal loci for variable-QoS mechanisms.

Although not universally accepted, *user-centered design* is an increasingly important paradigm in computer science. For example, it has emerged as the dominant approach in human-computer interaction and interface design [26], and as the basis for a large literature on Internet congestion control via pricing and related user feedback systems (e.g., [19, 29, 20, 23]). The fundamental premise is that a computer or networking system is only as good as its users believe it to be. When multiple objectives or various conflicting performance metrics are proposed, the design conflicts can be resolved by appealing to those choices that maximize some appropriate function of user valuations. User centricity not only provides a unifying approach to performance evaluation, but it also suggests a design principle: systems that are intelligently responsive to user expressions of relative value will tend to perform well.

In this paper we take a user-centered approach to the design and evaluation of Web caching replacement policies. We explore a scenario in which Web content providers (servers) reveal to a shared cache the value they receive from cache hits on their documents. We propose a hybrid

*MIT Workshop on Internet Service Quality Economics, Cambridge, Massachusetts, USA, 2–3 December 1999. Details available at `http://www.marengoresearch.com/isqe/`
Research funded by DARPA/ITO grant F30602-97-1-0228. Trace data courtesy NLANR (NSF grants NCR-9616602 and NCR-9521745). Jamin supported in part by NSF Career Award ANI-9734145, Presidential Early Career Award for Scientists and Engineers (PECASE) 1998, and equipment support from Sun Microsystems Inc. and Compaq (Digital Equipment) Corp. MacKie-Mason supported in part by an IBM University Partnership grant. The most recent version of this paper is available at `http://ai.eecs.umich.edu/~tpkelly/papers/`
`$Id: varp.tex,v 1.52 1999/11/13 04:09:11 tpkelly Exp $`

of the LFU (least frequently used) and LRU (least recently used) replacement policies that is sensitive to server valuations and that is tailored to the measured characteristics of Web client request patterns. Trace-driven simulations show that our algorithm sometimes outperforms other value-sensitive replacement policies, as measured by aggregate user value. We furthermore show that performance can be improved even more by tuning an aging parameter, and characterize conditions under which our algorithm does *not* work well.

When valuation declarations from servers influence cache removal priority, we might expect servers to strategically misreport their values in order to bias performance in their favor. Successful implementations of user-centered designs that involve contention over shared resources often require an *incentive mechanism* that induces truthful value reporting. Although the design of such a mechanism for our replacement policy is beyond the scope of this paper, it is an important problem, and one that will be shared by *any* user value-sensitive algorithm. We include a brief characterization of the economic incentives problem, and suggest directions in which it might be solved.

In the next Section we discuss the nature of value-sensitive replacement policies, and describe several from the existing Web caching literature. In Section 3 we explain how the traditional caching problem can be decomposed into two problems—value differentiation and prediction—and present several empirical analyses of Web trace data to justify the design choices we made for the prediction features of our algorithm. Section 4 presents empirical results comparing the value-sensitive performance of several value-sensitive algorithms. Section 5 describes circumstances under which biased frequency-sensitive algorithms such as ours do *not* perform well. Section 6 discusses incentive issues surrounding value-sensitive caching, and Section 7 concludes by surveying possible opportunities for generalizing existing value-sensitive cache management schemes.

2 Value-Sensitive Replacement Policies

Actual production caches currently employ LRU-like algorithms or periodic purge policies, often for reasons related to disk performance, but a far wider range of removal policies has been explored in the research literature. Williams et al. present a systematic taxonomy of policies organized by the sort keys that determine the removal order of cached documents [32]. For instance, LRU evicts documents in ascending order of last access time and LFU employs ascending reference count. See Bahn et al. [4] and the references therein for a recent review of the large literature on removal policies. The early literature on Web cache replacement algorithms considered policies intended to maximize performance metrics such as hit rate and byte hit rate; in a sense, the implicit design paradigm is one in which the cache designer "hard wires" into a cache the objective function it will maximize by specifying a fairly rigid replacement policy.

In recent years several research efforts have independently explored more flexible approaches to cache management. Many of these reflect a more sophisticated design approach in which a cache attempts to optimize an *arbitrary* objective function which is *not* built into the cache replacement policy. The need to provide different service levels to different content providers motivates our interest in such algorithms. We expect different servers to value cache hits on their objects differently, possibly with quite large differences. Some servers will have clients who are much less tolerant of delay, and who may be willing to pay for a higher quality of service. Some servers may be quite constrained in their own network connections and server equipment, and thus may value off-loading traffic to a network cache. The latter may especially make sense during temporary surges in demand for their objects that do not justify major capacity upgrades (e.g., following new software or document releases, or when a site otherwise becomes transiently hot, such as the NASA JPL site during the Jupiter probe fly-by). The existing market for object mirroring and distributed replication used by, e.g., software companies for distribution, is concrete evidence of the differing values that servers place on distributed object storage. Together with complementary research into variable-QoS Web content hosting [1], the small but growing family of value-sensitive caching policies address the needs of a heterogeneous user community.

2.1 Value Model

We assume that servers associate with each of their documents u a number W_u indicating the value they receive per byte when u is served from cache: the value generated by a cache hit equals $W_u \times \mathrm{size}_u$. This information can be conveyed to a shared cache in HTTP reply headers. We might speak of W_u as miss cost rather than hit value; the two perspectives are essentially equivalent. Thus, we can compare all replacement algorithms—value sensitive or insensitive, value or cost based—in terms of *value hit rate* (VHR), defined as

$$\mathrm{VHR} \equiv \frac{\Sigma_{\mathrm{hits}}\, W_u \times \mathrm{size}_u}{\Sigma_{\mathrm{requests}}\, W_u \times \mathrm{size}_u}. \qquad (1)$$

This performance metric is a natural generalization of the familiar byte hit rate measure: when all $W_u = c$ for some constant $c > 0$, VHR is equal to byte hit rate.

Several removal policies designed to maximize VHR have been proposed. Cao & Irani's "GreedyDual-Size" (GD-Size) algorithm attempts to optimize an arbitrary objective function that may be supplied dynamically, at cache run time [8]. Given weights W_u GD-Size seeks to maximize aggregate value (or minimize aggregate cost) across all requests. Following a request for u, the document's removal priority is set to $W_u + L$. L is an aging term initialized to zero; following a removal it is set to the priority of the evicted document. LRU breaks ties between documents whose removal priority is otherwise identical [7]. GD-Size is a value-sensitive *recentist* algorithm; when all values W_u are equal, it reduces to LRU.

Our original "server-weighted LFU" (swLFU) is a *frequentist* algorithm [14]. Removal priority is determined by weighted reference count $W_u \times N_u$, where N_u is the number of requests for u since it last entered the cache; last access time breaks ties between documents with identical value-weighted reference counts. When all W_u are equal and positive, swLFU reduces to LFU; when all weights are zero it becomes LRU. The algorithm retains those URLs that contribute most to aggregate user value per unit of cache space:

$$\frac{\text{contribution of } u \text{ to aggregate value}}{\text{unit size}}$$

$$= \frac{W_u \times \text{size}(u) \times N_u}{\text{size}(u)} = W_u \times N_u$$

Recently Arlitt et al. have introduced a frequency-sensitive variant of GD-Size, "GD-Size with Frequency" (GDSF) [3]. In GDSF a document's removal priority is set to $N_u \times W_u + L$ following a hit, where L has the same meaning as in GD-Size. Bahn et al. have developed a *family* of value-sensitive algorithms, collectively known as "Least Unified Value" (LUV), whose emphasis on frequency and recency can be adjusted [4]. Unfortunately we became aware of this work too late to include GDSF in all but one of our experiments, and we present no comparisons with LUV.

In this paper we compare value-sensitive replacement policies according to a value-sensitive metric. Before presenting our empirical studies, we suggest a conceptual framework in which to analyze caching policies. Using this framework, we then uncover regularities in trace data that guide our algorithm design.

3 Prediction vs. Value Sensitivity

One approach to designing Web caching systems, typical of much of the earliest literature, is to implement new features on an ad hoc basis and test performance experimentally. A more refined approach, increasingly common in recent work, is to identify regularities in Web cache workloads and to implement features that are well-suited to these regularities (see Reference [27] for a sophisticated example). To put this latter approach on a solid basis, we suggest a conceptual framework within which to analyze trace data and to design caching policies. We then present empirical analysis within this framework that guides our design choices.

The performance of any user-value-sensitive caching system depends on how well it solves two distinct problems: *prediction* and *value differentiation*. Any measure of performance will depend on having objects already waiting in the cache before they are requested, hence prediction. Since resources (network bandwidth, CPU, disk space, human time for management) are scarce and costly, and objects are created and changed in real time, we cannot always have *all* objects waiting in cache in advance. Therefore, of the set of objects predicted to be requested, we need to differentiate their value to determine *which* to cache.

Value-insensitive algorithms have largely focused on solving the prediction problem, ranking documents for removal based on estimated likelihood of future requests. Thus, we might expect recentist algorithms like LRU to perform well when there is substantial temporal locality in user requests; frequentist approaches are better suited to time-independent requests.

In this paper, we focus primarily on the relatively new problem of value differentiation. However, an algorithm will not serve users well if it does value differentiation well, but performs poorly at prediction. Therefore we analyzed trace data and the prior literature to find regularities important for *prediction*, and used these findings to hardwire certain features into our algorithm, while allowing value differentiation to be dynamically driven by user valuations. Our value/prediction framework is similar in spirit to an elegant approach developed independently by Bahn et al. [4], though perhaps different in emphasis.

From our data and the prior literature, we have identified four Web request stream characteristics relevant to *prediction*:

1. Low temporal locality of reference.
2. Zipf-like document popularity distribution.
3. Nonstationary request process.
4. Weak size-popularity correlation.

We measure temporal locality in a request stream with an LRU stack distance transformation. We add the items in the stream to an infinite-capacity stack as follows: if the item is not present in the stack, we push it on the top (at depth 1) and output "miss"; this increases by 1 the depth of all items already in the stack. If an item *is* present in the stack ("hit"), we output its depth, remove it, and replace it at the top. For example, the symbol stream "ABBACBD" yields "miss miss 1 2 miss 3 miss." Maximal temporal locality occurs when all references to the same symbol are adjacent on the input, in which case all hits occur at depth 1; the string "AABBBCD" has the same relative symbol frequencies as in the previous example, but now a stack distance transform yields "miss 1 miss 1 1 miss miss." See References [5] and [21] for a more detailed explanation of the stack distance model and its application to Web caching.

Figure 1 shows the CDF of LRU stack hits in 14-day

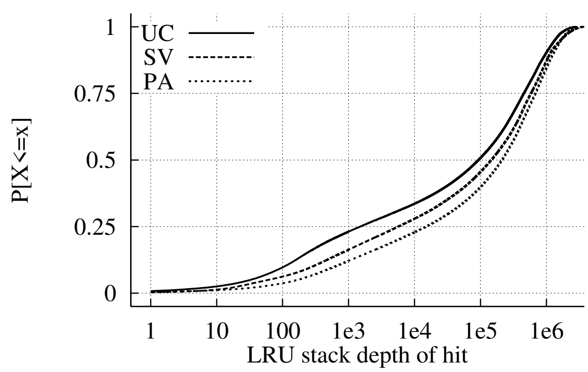

Figure 1: CDF of LRU stack distances of hits.

request streams collected during August 1998 at three NLANR caches [12]. These traces range in length from 5.5 million to 7.9 million requests; see Reference [14] for details. The median stack depth of hits ranges from 100,000 to 200,000, indicating weak temporal locality. This conclusion is consistent with several recent findings, e.g., Mahanti & Williamson, who report consistently low temporal locality across several shared-cache workloads [21], and Barford et al., who report that temporal locality in *client* traces declined between 1995 and 1998 [5]. The implication is that pure recentist algorithms like LRU may not have very high predictive success.

Our second observation is that the frequency of document requests in our traces is Zipf-like, i.e., the number of references to the ith most popular document is proportional to $1/i^{\alpha}$. This is qualitatively apparent in Figure 2, a log-log plot of reference count as a function of popularity rank for six 28-day NLANR traces collected during March 1999; Table 1 presents estimates of the α parameter. If we assume that temporal locality is so weak as to be negligible and that document references are independent, the Zipf-like popularity distribution argues strongly in favor of frequentist prediction; see Breslau et al. for a more thorough discussion [6].

Even if document references are independent, the distribution that generates them may change over time. This

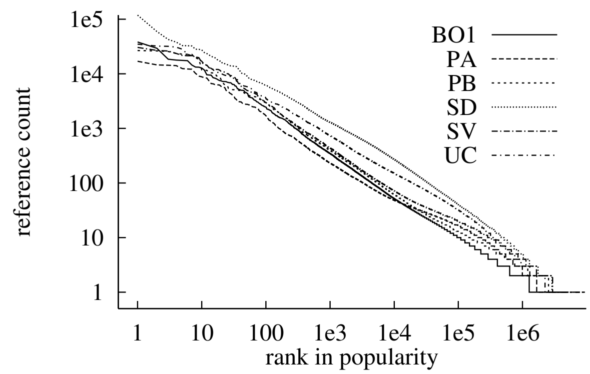

Figure 2: Zipf-like popularity distribution in March 1999 traces.

effect is apparent when we examine day-to-day changes in the set of popular documents ("hot set drift"). Figure 3 shows, for each of the first 28 days in March 1999

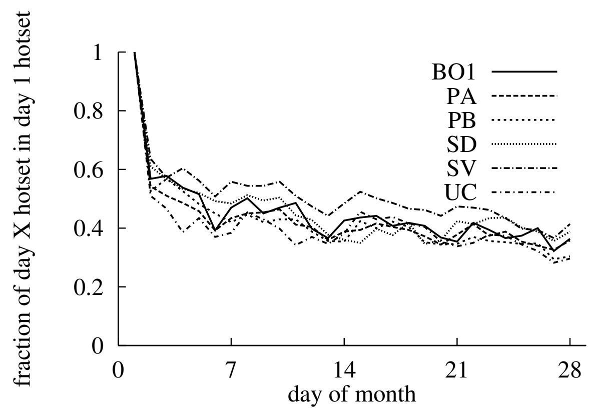

Figure 3: Hot set drift at six NLANR sites, March 1999.

at each of six NLANR cache sites, the fraction of that day's 500 most popular documents that were among the 500 most popular on 1 March. We see that the composition of the "hot set" changes gradually over time (Mahanti & Williamson report qualitatively similar results for other traces [21]). The implication is that pure frequentist prediction (LFU) will likely suffer a "cache pollution" problem: formerly-popular documents that are no longer requested often will clutter the cache as time goes on. We confirmed the pollution conjecture by a simple experiment: using one of our August 1998 NLANR traces, we simulated 4GB and 8GB caches using LRU and LFU. We compute hit rates separately within non-overlapping windows of 250,000 requests each, shown in Figure 4. LFU initially outperforms LRU, but over time its performance deteriorates; the problem is more severe at the smaller cache size. Thus, in time series terms, we have observed low positive serial correlation at high frequencies

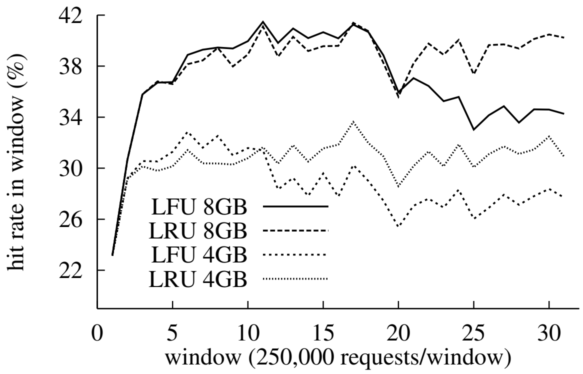

Figure 4: LFU cache pollution: windowed hit rates for LRU and LFU at two cache sizes, August 1998 SV trace.

(absence of temporal locality), and negative serial correlation at low frequencies (hot set drift). Recency information can play an important role in prediction, but policies like LRU do not exploit it well.

Finally, no clear relationship between document size and popularity is evident in the six traces used in our experiments. In Table 1 we provide summary statistics on the six traces we use, including size-frequency correlations. In each trace the correlation between document size and popularity does not differ significantly from zero. The design implication is that we should not discriminate against either large or small documents.

From our analysis of request streams we conclude that a mix of frequentist and recentist prediction is appropriate. We implement a simple convex combination of LFU and LRU, together with value sensitivity, for an algorithm we call *aged server-weighted LFU* (A-swLFU). The default replacement policy is to evict objects based on the value-weighted frequency count as described in Section 2; however, on every Kth eviction we remove the LRU item. This reduces to original swLFU and plain LRU as special cases ($K = 0$ and $K = 1$, respectively). We are not the first to explore recentist/frequentist hybrids: Lee et al. [18] define a different continuum between LRU and LFU for the *unweighted* case ($W_u = 1$ for all u). Bahn et al. have recently generalized this algorithm to the *weighted* case of interest to us [4]. Unfortunately we became aware of this work too late to include it in our experiments.

Adjustable parameters may impose a burden on administrators if they are to be well-tuned, and thus need to be justified. Our K-aging has a nice property, however: it is essentially an optional increment over current algorithms. The choice of K could be hard-wired; indeed the choices 0 and 1 are equivalent to well-known pure frequentist and recentist approaches. Since we know that a K other than 0 or 1 can substantially improve performance, cache administrators can make the decision whether the benefits are sufficient to justify the additional burden of parameter tuning.

Whereas "LRU" unambiguously specifies a replacement policy, the family of LFU-like algorithms are parameterized by answers to the following questions:

1. What criteria break ties between documents with identical reference counts?

2. Are reference counts maintained on items even after they have been evicted from cache?

3. After a document request is processed, is the document *guaranteed* to be in cache? (Is placement following a miss mandatory or optional?)

Figure 5 shows the impact of the last two parameters on byte hit rate for LFU algorithms that use LRU to break ties. Throughout this paper we use LRU as a secondary

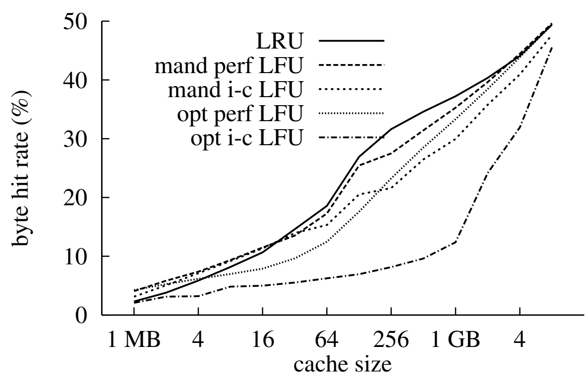

Figure 5: Byte HR at cache sizes 1MB–8GB for LRU and four LFU variants, August 1998 NLANR SV trace.

removal criterion in all algorithms. We explore variants of LFU in which reference counts persist across evictions ("Perfect LFU" in the terminology of Breslau et al. [6]), and in which they are defined only for cached items ("in-cache LFU"). While some theoretical investigations consider the case of optional placement [13], we find empirically that it *never* confers a substantial advantage over mandatory placement and often incurs a severe performance penalty, possibly because it ensures that a large fraction of the many twice-requested documents in our traces never result in cache hits. Therefore we consider only mandatory-placement variants of LFU.

4 Experiments

We compare value-sensitive removal policies through trace-driven simulations using Web request streams collected at six NLANR cache sites during 1–28 March

Table 1: Traces recorded at six NLANR sites, 1–28 March 1999.

	BO1	PA	PB	SD	SV	UC
requests	11,583,087	13,548,917	19,803,754	37,085,277	23,738,274	26,024,662
documents	5,252,946	4,901,241	9,820,054	8,640,338	9,375,514	7,615,462
servers	193,422	168,082	291,410	247,459	265,305	250,484
unique bytes	104,474,161,664	76,038,927,331	188,308,442,149	204,928,271,675	159,114,665,878	150,119,984,279
bytes requested	236,150,085,697	220,658,618,173	383,130,815,921	620,283,701,022	412,899,064,992	397,548,684,913
max H.R. (%)	54.6	63.8	50.4	76.7	60.5	70.7
max B.H.R. (%)	59.4	67.3	55.2	69.1	63.2	64.4
mean N_u	2.205	2.764	2.017	2.532	4.292	3.417
std. dev. N_u	37.77	25.60	29.71	34.10	74.59	43.80
Zipf α (R^2)	.578 (.88)	.751 (.93)	.560 (.89)	.854 (.93)	.692 (.92)	.784 (.91)
mean size_u	19,888.68	15,514.22	19,175.91	16,971.30	23,717.62	19,712.52
std. dev. size_u	337,424.3	220,990.6	269,819.6	221,649.6	289,507.6	312,418.1
median size_u	3895	3584	3712	3886	4080	3830
Cov(size_u, N_u)	5158.76	4533.34	4168.46	-25,014.49	3097.10	-11,984.42
Corr(size_u, N_u)	0.00040476	0.00080136	0.00052007	-0.00115845	0.00040978	-0.00087585

1999 [12]. Prior to simulation we pre-process raw cache access logs by removing dynamic content and preserving only successful requests for items not present in client caches. Our six processed traces are summarized in Table 1.

4.1 Heterogeneous Valuations

To explore the relative performance of value-sensitive removal policies, we conducted experiments of the following form: Randomly assign to each server s a weight W_s drawn uniformly from the set $\{1, 10, 100, 1000, 10000\}$, then set $W_u = W_s$ for all documents u hosted by server s, and finally compute value hit rates for various algorithms at different cache sizes. We use a a high-variance weight distribution because, as Section 5 explains in greater detail, weighted-LFU algorithms behave very much like ordinary unweighted LFU when weights span a narrow range. Intuitively, weight-sensitive algorithms aren't very helpful in the relatively uninteresting case when all weights are similar. In Figure 6 we show mean VHR over five weight assignments at cache sizes ranging from 64MB to 16GB for perfect and in-cache variants of A-swLFU with $K = 100$; no attempt was made to tune K to particular traces or cache sizes. We present LRU at cache sizes from 1–16GB to illustrate the gap between conventional and value-sensitive algorithms. Our results suggest that even without a well-tuned aging parameter A-swLFU offers negligible performance advantages over GD-Size at larger cache sizes (4-16GB), but consistently yields better VHR at smaller cache sizes in most of our traces.

In Figure 7 we show the potential gains from tuning the K parameter. We computed VHR averaged over 20 random assignments of W_u for GD-Size, in-cache GDSF, and perfect and in-cache A-swLFU with K values of $0, 10, 20, \ldots, 150$; at each cache size we present the A-swLFU with the highest VHR. Perfect A-swLFU per-

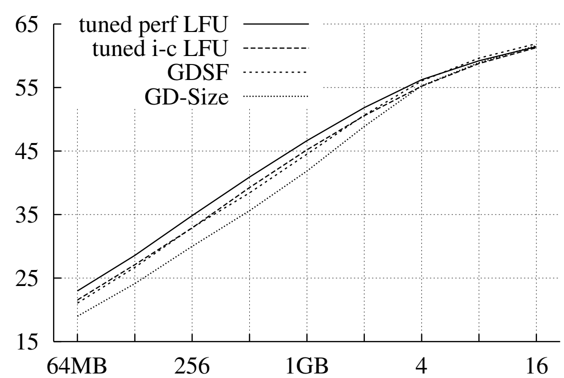

Figure 7: Tuned perfect & in-cache A-swLFU, in-cache GDSF, and GD-Size. March 1999 UC trace.

forms the best for caches that are 4GB or smaller, consistent with what Breslau et al. predict and empirically find in the *unweighed* case where all weights are 1 and the performance metric is byte hit rate [6]. However the gains over in-cache A-swLFU and GDSF are modest and may not justify the extra cost of retaining frequency tables on evicted documents. Optimally-tuned in-cache A-swLFU and GDSF perform almost identically; since both are value-sensitive combinations of recentist and frequentist approaches, this is not surprising. GD-Size, which does not exploit frequency information, performs noticeably worse except at large cache sizes. Figure 9 and the accompanying text in Section 4.2 discuss tuning the K parameter in greater detail.

Aged weighted LFU appears to work best when cache space is moderately scarce. This performance advantage might be especially important if main-memory caches become common. At least some current caching systems appear to be disk I/O constrained [28]. It is conceivable that Web demand and network bandwidth will grow so rapidly that disk bandwidth cannot keep pace, in which

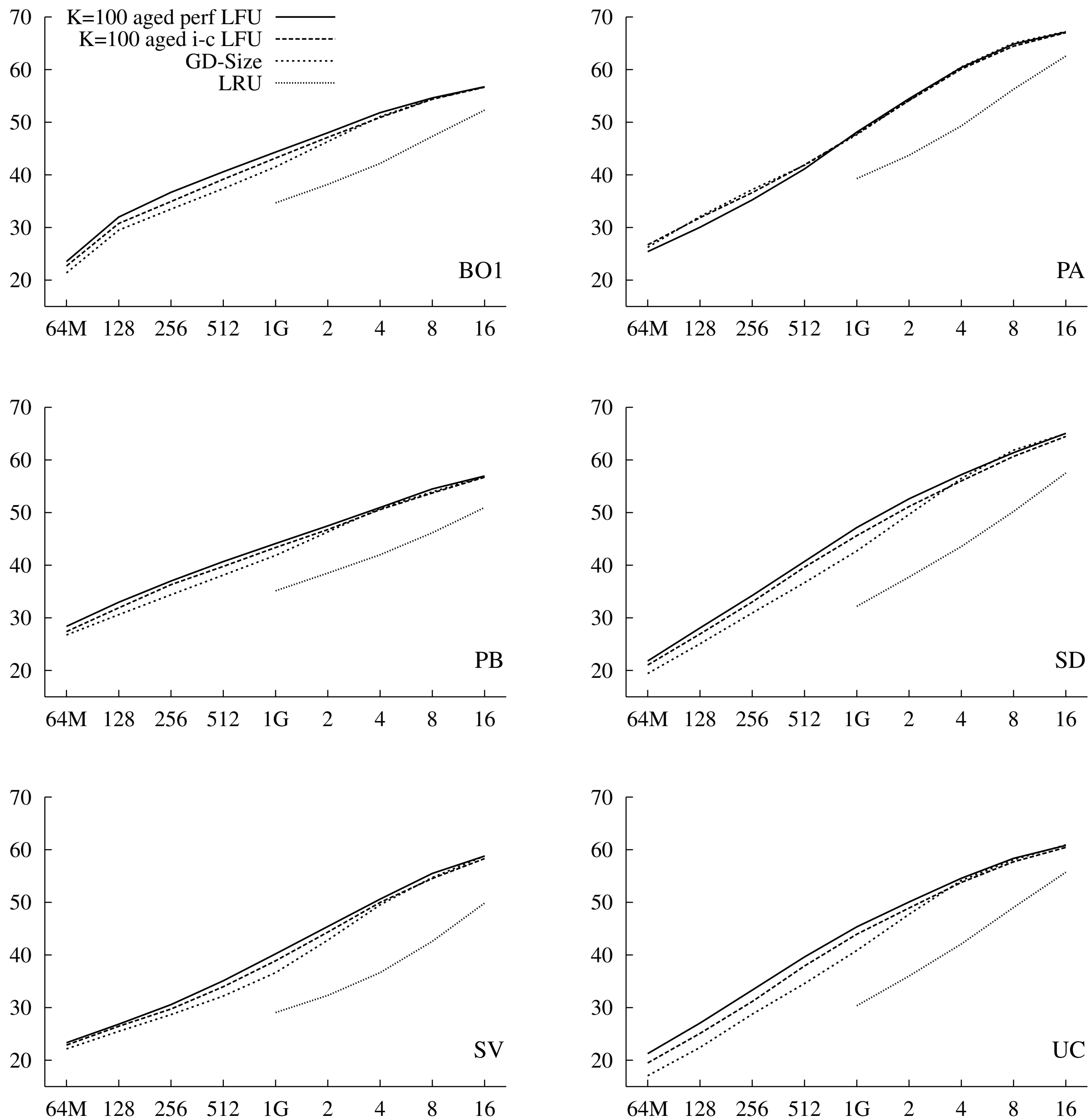

Figure 6: VHR as function of cache size for two A-swLFU variants and GD-Size. LRU also shown at larger cache sizes for comparison.

case RAM-only caches would become a reasonable design option. The absence of disks would remove many practical constraints that currently limit cache designers' choice of removal policy. A value-sensitive replacement algorithm would enable a modest-sized diskless cache to provide "premium" service for those willing to pay for minimal latency. Our results show that GDSF and A-swLFU are good replacement policies for such a cache.

4.2 Homogeneous Valuations

As a "sanity check" we also consider the degenerate case where all documents have equal weight, $W_u = 1$ for all u.[1] As noted in Section 2, GD-Size reduces to ordinary LRU in this case, and our VHR performance metric reduces to byte hit rate. Figure 8 presents byte hit rates at cache sizes ranging up to 16GB generated by GD-Size/LRU and four LFU variants (all combinations of aged ($K = 10$) vs. ordinary ($K = 0$) and perfect vs. in-cache). Our results confirm Breslau et al.'s conclusion that (un-aged) in-cache LFU performs poorly in terms of byte hit rate [6]. However, we find that the addition of aging *without* any attempt to tune the aging parameter improves the performance of in-cache LFU beyond that of un-aged perfect LFU. As expected, aged perfect LFU generally performs best. Finally, in three of six cases (PA, PB, and SD) LRU outperforms un-aged perfect LFU at all cache sizes, contrary to Breslau et al.'s claim that perfect LFU generally performs better than LRU in terms of BHR. We attribute the difference to the size of Breslau et al.'s traces, which are too small for the cache pollution effect we see in Figure 4 to affect LFU. More remarkably, aged in-cache LFU outperforms aged perfect LFU on two traces (PA and SD), and performs roughly as well one other (SV).

How much can we potentially gain by tuning K at a particular cache? Figure 9 shows byte hit rate as K varies from zero to 25 for in-cache LFU (solid lines) and perfect LFU (dashed lines) at cache sizes ranging from 256MB (lowermost solid/dashed pair) to 16 GB (top pair). (The solid and dashed lines meet at $K = 1$ because both algorithms reduce to LRU at that K value.) Remarkably, at *every* cache size in-cache LFU with optimal K outperforms perfect LFU with optimal K. In other words, it appears that well-tuned aging might eliminate any advantage of maintaining reference counts on evicted documents in the heterogeneous valuation (unweighted) case. Figure 9 furthermore appears to confirm our earlier conjecture that the optimal amount of aging depends on cache size; larger caches require more aggressive aging (lower K).

[1] GDSF appeared after we conducted this series of experiments and could not be included in this comparison.

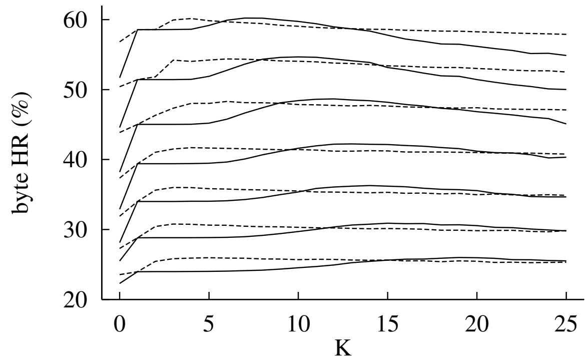

Figure 9: Byte HR as function of aging parameter K for in-cache LFU (solid lines) and Perfect LFU (dashed lines). March 1999 SD trace.

5 Limits to Biased LFU

We can identify at least two situations in which weighted-LFU algorithms do not perform much better than their unweighted counterparts: when value differences are undone by the law of large numbers, and when weights span a narrow range.

To see the first point, consider a *client-weighted* variant "cwLFU" in which client i supplies weight w_i indicating the utility per byte it receives when its requests are served from cache. Removal priority in cwLFU is determined by

$$V_u \equiv \sum_{\text{clients } i} w_i n_{iu}$$

where n_{iu} is the number of requests for URL u by client i. The problem with this approach is that when client weights w_i are uncorrelated with reference counts n_{iu}, the law of large numbers causes the quantity

$$\overline{V}_u \equiv \frac{V_u}{N_u} \quad \text{where} \quad N_u \equiv \sum_i n_{iu}$$

to converge toward the mean of the distribution from which the w_i are drawn for URLs with high overall reference counts. If weights are uniform over $\{1, 2, \ldots, 10\}$, for instance, popular URLs will tend to have $\overline{V}_u$ close to 5.5. Ordinary LFU and cwLFU differ only insofar as $\overline{V}_u$ differ substantially across objects, and this does not happen when client weights are uncorrelated with reference counts. It is conceivable that such correlations do exist in the real world, e.g., we might imagine that impatient clients who value cache hits highly have similar reading habits. However, such correlations are difficult to model and we do not speculate further about them.

We have also found that swLFU does not perform well with weights drawn from a narrow range, e.g., 1–10. The

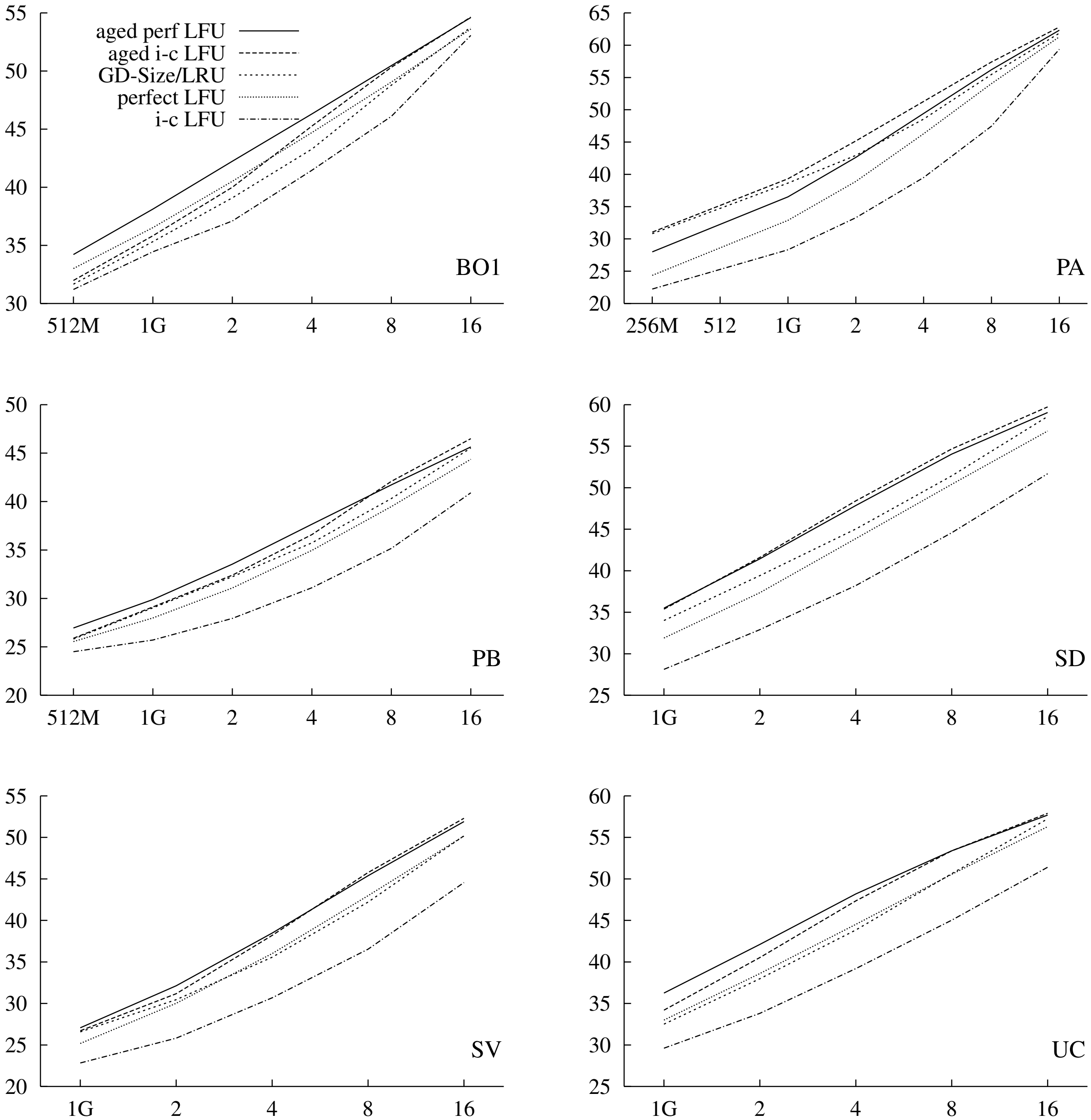

Figure 8: Cost = size case: byte hit rates as function of cache size for GD-Size/LRU and four LFU variants: perfect vs. in-cache and K=10 aging vs. no aging. March 1999 NLANR traces. Note that vertical scales vary.

reason is that URL reference counts N_u vary over many orders of magnitude (Figure 2). If weights W_u span only one order of magnitude, their influence on the behavior of swLFU may be negligible.

We can illustrate both of weighted LFU's difficulties through a simple experiment: obtain URL reference counts n_{iu} from an actual Web cache access log, and assign to the clients in the log weights w_i drawn randomly from $\{1, 2, \ldots, 10\}$. Create two lists of URL tuples of the form (u, N_u, V_u), one sorted in descending order of reference counts N_u and the other sorted on cwLFU removal priority V_u. Examine the overlap in the top k URLs on both lists as a function of k. If the two lists are very similar, the top k sub-lists will overlap substantially even for small values of k; if the lists are very different, the overlap will be small except for large values of k. This exercise provides a crude way to compare the contents of weighted and unweighted caches: the top k items on our two sorted lists are roughly those that would be contained in unweighted LFU and cwLFU caches of size k after processing the request stream in the access log. This experiment can be performed for swLFU as well as cwLFU; in both cases removal priority is weighted reference count. Figure 10 shows list overlap as a function of k in two scenarios: client weights drawn from a narrow ranges (top), and server weights drawn from our high-variance distribution (bottom). Reference counts n_{iu} are from the NLANR SV log of 17 March 1999. For a cache capable of holding between 10,000 and 100,000 documents, weighted and unweighted LFU yield very similar cache contents (80% overlap), and therefore similar hit/miss behavior, in the narrow-weight-range cwLFU case. By contrast, the similarity between weighted and unweighted cache contents is far lower (25% overlap) in the wide-weight-range swLFU case.

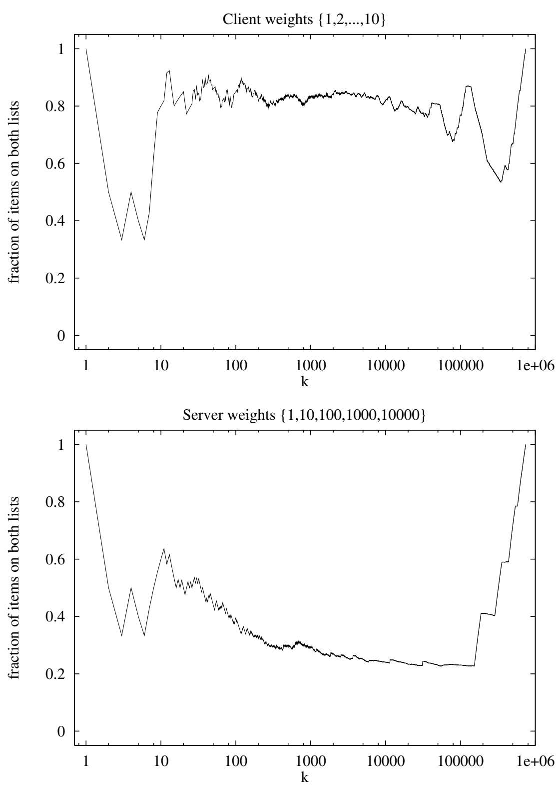

Figure 10: Overlap among top k items in lists sorted on weighted and unweighted criteria.

6 Incentives

User-centric value-sensitive replacement policies require information about user valuations. By measuring performance (VHR) using server announcements of their values (W_u), we have been implicitly assuming that these announcements are truthful. Unfortunately, when cache replacement is directly affected by the announced values, it will generally be in each server's private interest to systematically misreport its valuations: no matter how low their true values, they would like their objects to get better treatment than another server's objects. The problem of strategic announcements is generic and confronts any value-sensitive replacement policy: a reliable source of user value information is needed to improve on insensitive policies.[2]

A powerful approach to this problem is known as *mechanism design*; see Reference [22] for a good introduction. The approach is to provide participants with economic incentives such that it is in their rational self-interest to provide useful valuation information. The search over possible incentive schemes is considerably simplified by the Revelation Principle [25], which states that any aggregate user value that can be achieved by some incentive scheme can be equivalently achieved by a scheme in which it is rational for participants to tell the truth. Nonetheless, the design of incentive mechanisms is technically challenging, and is beyond the scope of this paper. We merely offer some observations on the possible shape of a good scheme.

One important result originally due to Vickrey [31] and generalized to a much richer set of problems in Reference [30] lends some intuition for the problem. Vickrey proposed the second price auction: charge the winner of a single good auction the second highest bid. The bidder's announcement affects only *when* she wins, not how much she pays, and it can be shown that the bidder's dominant

[2]The problem of inducing servers to truthfully reveal private valuation information is distinct from the problem of preventing a *cache* from over-reporting hits in a scheme in which servers pay for cache hits. Economics offers insight into the former problem ("bid shading"), but not the latter (fraud).

strategy is to bid her true valuation for the good being sold.

The Varian & MacKie-Mason generalization [30] suggests that charging a server for each hit the valuation announced for the object that was most recently evicted might be incentive compatible. This would work if caching decisions were a one-shot activity. Unfortunately it is not, and in this example, the server's bid would affect *future* payments, and thus it will not be optimal to tell the truth. For example, if the current price is less than the server's true value, it will want to overbid in order to increase its object's duration in the cache, since each hit will produce value greater than its cost.

In another paper we proposed a quite different approach to value-sensitive caching, in which a cache periodically auctions off disk space [9]. In that setting we were able to provide an incentive-compatible scheme.

7 Lessons Learned

The research on value-sensitive approaches to network transport priorities and guarantees (for admission and service) continues to be an extremely active area.[3] That user demand for variable QoS is substantial seems evident from the wide variety of prices that users pay for Internet connections with varying bandwidth. Policies that can be implemented within the network to allocate scarce resources offer the possibility of greater flexibility and dynamism, and thus the opportunity to increase the aggregate value of the network to its users. Policies that offer greater degrees of (possibly stochastic) QoS guarantees also would support the widespread deployment of inelastic, latency-sensitive applications such as real-time audio and video.

One of the main points of our research is that network admission and transport are not the only scarce resources that can be managed to offer variable QoS. For any given topology of servers, links and switches, and any given allocation of bandwidth and transport priorities, QoS for object delivery (e.g., Web usage) will be affected by the location of servers and their local congestion conditions. Therefore, the level and variability of network latency will be affected by the topology of object storage. Our research is part of a much smaller QoS literature that has recently begun to consider the use of value-sensitive approaches to distributed network file storage as a means to improve network value through user-responsive variable QoS.

We have focused on replacement policies at a single L3 Web cache as one target for value-sensitive file storage. A few other recent papers have also explored value-sensitive replacement policies for aggregator Web caches. In particular, Cao & Irani introduced the first explicit value-sensitive approach of which we are aware [8], and Arlitt et al. suggest a variant that performs slightly better than our A-swLFU without requiring a tunable aging parameter. However, in our view the literature is far too immature to focus on horse races between specific algorithms; the important result is that value-sensitive algorithms deliver substantially higher value to the user community than their insensitive counterparts.

Our results underscore the importance of workload characteristics for the effectiveness of value-sensitive Web caching algorithms. The temporal and spatial distribution object requests can have first-order effects on the potential gain from any value-sensitive replacement policy over an insensitive policy. Further, effective design within the class of value-sensitive policies is likely to be driven by workload characteristics (e.g., the importance of an aging mechanism for frequentist replacement policies). Interestingly, the network transport QoS pricing literature has not been uniformly attentive to the empirical characteristics of offered traffic. However, it is worth noting for any value-sensitive QoS policy research, whether directed to network transport, file storage or some other network resource, that users are responsive and adaptive agents, and that workloads observed under unpriced conditions will change if prices are introduced. Thus, a critical need for all such research is to develop more data on the responsiveness of users (at various levels of aggregation) to quality-sensitive pricing for network resource usage.

A closely related lesson for Web caching replacement policies—value-sensitive or otherwise—is that we need to understand the interaction between usage aggregation and cache allocation. We have studied caching at the L3 level of the NLANR shared cache hierarchy: that is, for caches that are designed to serve only L2 organizational shared caches, which in turn serve individual workstation caches. Obviously, the organizational hierarchy of shared caches will have dramatic effects on workload distributions. For example, if a user at the University of Michigan requests a cacheable document, it will be cached at the L2 cache serving UM. Thereafter, until the document is removed, future requests for it from users at UM will be served by the UM L2 cache. In other words, within any reasonable time horizon, the L3 cache should only see a single request for any cacheable object from a given L2 cache. If an L3 cache typically serves twenty L2 caches, then it would be unusual to see more than 20 requests for a single object during a typical time window. Indeed, the median number of requests for objects in our datasets was one, which drastically limits what even the most clever cache

[3] For just a few recent theoretical and empirical examples, see, e.g., References [10, 2, 16, 15, 17], and indeed most of the papers at the MIT Workshop on Internet Service Quality Economics, December 2-3, 1999, Cambridge, MA.

replacement policy can accomplish.[4] On the other hand, the L2 cache at Michigan serves approximately 80,000 active user accounts, and might see thousands of requests for a popular object. Obviously, workload characterizations will be quite different at different locations in a shared caching topology. But, since the topology is also a designed feature, caching to implement variable QoS needs to consider simultaneously the replacement policies at individual caches *and* the design of the sharing topology.

The interaction between topology and replacement algorithms highlights another important question for further research: value-sensitive inter-cache allocations. For example, within a cooperative set of shared caches, it is necessary to have policies to determine which caches will store which objects, and which caches (or lower-level users) are permitted to retrieve those objects. Based on our experience with workload characteristics and the strong interactions between sharing topology and workload, we think there may be significant gains from value-sensitive, dynamic sharing protocols rather than static hierarchies. Price or value messages between caches would typically provide an efficient, terse summary of the information necessary to implement more efficient distributed caching. The gains from a more adaptive, responsive storage topology could well be greater than from minor improvements in single-cache replacement policy algorithms. Therefore incorporating value concerns into existing inter-cache protocols appears to be a promising direction for future research.

We also envision the possibility that noncooperative caching servers might be simultaneously active within the network; indeed, this is already true in today's commercial Internet. In a market setting with noncooperative, self-interested cache managers competing to provide QoS, the case for value-sensitive inter-cache protocols and replacement policies is quite natural. For example, when a given cache manager finds that one of its regional caches is becoming dangerously congested, it might find it more effective to temporarily rent caching space or priority on a nearby independent server rather than route client requests back to the object owner's originating server.

Extending the scope of the problem yet another step, it should be evident that there is not necessarily anything that limits value-sensitive allocation policies to Web caches (or network links). By recognizing that the entire range of network resources involved in object storage, manipulation and transport have an effect on the user's perceived QoS, we open the possibility that other resources might be usefully guided by value-sensitive allocation policies. The obvious extension to our work—and another area in which we expect the net gains may be greater than from continued marginal improvements in single Web cache replacement policies—is to consider value-sensitive protocols and allocation mechanisms for a greater range of distributed file storage systems. One application might be the sharing of unused workstation disk space to provide a distributed LAN file storage server; Douceur & Bolosky's study of disk usage on a large corporate network indicates that roughly half of all disk space is unused [11], so the potential gains from tapping this resource are large. Likewise, at the LAN or Internet level value-sensitive protocols might be able to implement efficient distributed file backup systems. A recent application of considerable interest is the provision of network file storage for roaming or mobile users [24].

One very important issue for any value-sensitive scheme, as we discussed in the previous section, is how to obtain information about users valuations. In systems with autonomous agents it is generally true that at least some user valuation information is private and users cannot be compelled to truthfully reveal that information. This provides quite a challenge to resource allocation approaches that try to use valuation information in order to achieve the sensible goal of maximizing aggregate value. We have learned from mechanism design theory that it is quite difficult, and in some seemingly simple situations provably impossible, to design allocation mechanisms—market-based or otherwise—that induce value revelation in such a way as to permit allocations satisfying reasonable requirements, e.g., voluntary participation or budget balance. Such mechanism design problems are especially pernicious in situations with costly computation and repeated interactions.

However, we do not mean to suggest too much pessimism. In the simple case of single-cache replacement, value-sensitive replacement policies have the potential to improve performance substantially compared with value-insensitive policies. Therefore, even if information revelation incentives can induce only approximately complete revelation, the QoS gains may be worth pursuing. The additional gains potentially available from taking into account interactions with the storage topology, inter-cache transactions, and other types of file storage strengthen the case for research on market-based (and perhaps other) incentive schemes.

Indeed, the significance of information sharing points to a set of distributed file system policies that have received almost no attention thus far: service, as opposed to replacement, policies. Replacement policies determine what objects are placed and or retained in a particular file store. Service policies determine which objects are delivered to users from the store, and with what service qual-

[4] The observant reader might have noticed from our Figure 2 that we appear to have at least 1000 objects for which there are an implausibly large number of requests. Our NLANR log files do not allow us to identify and remove uncacheable content prior to simulation experiments, and we suspect that many of the documents in our filtered traces are actually uncacheable. We thank John Dilley and Duane Wessels for helping to bring this anomaly to our attention.

ity. Pricing for packet scheduling priority is an example: when a user pays a particular price, it is granted a particular level of service (priority). Likewise for file storage: Users (possibly aggregators or lower level caches) or competing caches might pay different prices to obtain different levels of file delivery service. The payments and the service quality parameters can be adjusted to induce users to reveal information about their valuations for file service QoS.

To summarize, our work on value-sensitive single-cache replacement policies teaches several important lessons for the design and implementation of QoS-enabling allocation policies in the Internet. Those lessons apply to the design of specific algorithms such as ours for Web caching, and also guide us towards other research problems that seem likely to be at least as rewarding.

Acknowledgments

Professor Michael Wellman provided valuable guidance to this project. We thank Jonathan Womer and Yee Man Chan for many interesting discussions. Duane Wessels of NLANR helped us to understand several oddities in our trace data. Tom Hacker, the University of Michigan Center for Parallel Computing, and Jeff Kopmanis made available computational resources essential to our empirical work. Finally we thank Martin Arlitt and John Dilley of HP Labs for an interesting exchange of correspondence.

References

[1] Jussara Almeida, Mihaela Dabu, Anand Manikutty, and Pei Cao. Providing differentiated levels of service in Web content hosting. In *Proceedings of the ACM SIGMETRICS Workshop on Internet Server Performance (WISP)*, 1998.

[2] Jörn Altmann, Björn Rupp, and Pravin Varaiya. Internet demand under different pricing schemes. In *Proceedings of the ACM Conference on Electronic Commerce (EC'99), Denver, CO*, November 1999.

[3] Martin Arlitt, Ludmila Cherkasova, John Dilley, Richard Friedrich, and Tai Jin. Evaluating content management techniques for Web proxy caches. In *Proceedings of the Second Workshop on Internet Server Performance (WISP '99)*, May 1999. Also available as an HP Labs tech report at `http://www.hpl.hp.com/techreports/98/HPL-98-173.html`.

[4] Hyokyung Bahn, Sam H. Noh, Kern Koh, and Sang Lyul Min. Using full reference history for efficient document replacement in Web caches. In *Proceedings of the Second USENIX Symposium on Internet Technologies and Systems (USITS99)*, pages 187–196, November 1999. `http://www.cs.hongik.ac.kr/~dnps/research/pub.html`.

[5] Paul Barford, Azer Bestavros, Adam Bradley, and Mark Crovella. Changes in web client access patterns: Characteristics and caching implications. *World Wide Web Journal, Special Issue on Characterization and Performance Evaluation*, 1999. Also available as Boston U. CS tech report 1998-023 at `http://www.cs.bu.edu/techreports/`.

[6] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. Web caching and Zipf-like distributions: Evidence and implications. In *Proceedings of IEEE Infocom99*, March 1999. Tech report version available at `http://www.cs.wisc.edu/~cao/papers/`.

[7] Pei Cao and Sandy Irani. Personal communication.

[8] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems*, pages 193–206, December 1997. `http://www.cs.wisc.edu/~cao/papers/gd-size.html`.

[9] Yee Man Chan, Jeffrey K. MacKie-Mason, Jonathan Womer, and Sugih Jamin. One size doesn't fit all: Improving network QoS through preference-driven Web caching. In *Proceedings of the Second Berlin Internet Economics Workshop*, May 1999.

[10] Kai Cieliebak and Beat Liver. How many QoS classes are optimal? In *Proceedings of the ACM Conference on Electronic Commerce (EC'99), Denver, CO*, November 1999.

[11] John R. Douceur and William J. Bolosky. A large-scale study of file-system contents. In *Proceedings of ACM SIGMETRICS*, 1999. Cool study of filesystems at MS.

[12] National Laboratory for Applied Network Research. Anonymized access logs. `ftp://ftp.ircache.net/Traces/`.

[13] Sandy Irani. Page replacement with multi-size pages and applications to Web caching. In *29th ACM STOC*, pages 701–710, May 1997.

[14] Terence Kelly, Yee Man Chan, Sugih Jamin, and Jeffrey K. MacKie-Mason. Biased replacement policies for Web caches: Differential quality-of-service and aggregate user value. In *Fourth International Web Caching Workshop*, March 1999. `http://ai.eecs.umich.edu/~tpkelly/papers/wlfu.ps`.

[15] Peter Key and Laurent Massoulie. User policies in a network implementing congestion pricing. In *MIT Workshop on Internet Service Quality Economics, Cambridge, MA*, December 1999.

[16] K. Kumaran, M. Mandjes, D. Mitra, and I. Saniee. Pricing of variable bit rate services based on trace-based leaky bucket parameter estimation. In *MIT Workshop on Internet Service Quality Economics, Cambridge, MA*, December 1999.

[17] Steve Lanning, William A. Massey, Brian Rider, and Qiong Wang. Pricing of IP services with quality of service constraints. In *MIT Workshop on Internet Service Quality Economics, Cambridge, MA*, December 1999.

[18] Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. On the existence of a spectrum of policies that subsumes the LRU and LFU policies. In *Proceedings of the 1999 ACM SIGMETRICS Conference*, pages 134–143, 1999. `http://www.cs.hongik.ac.kr/~dnps/research/e99/99sigmetrics-ps.ps`.

[19] Jeff MacKie-Mason and Hal Varian. Pricing congestible resources. *Journal of Selected Areas in Communications*, 13(7):1141–1149, September 1995.

[20] Jeffrey K. MacKie-Mason, Liam Murphy, and John Murphy. The role of responsive pricing in the internet. In Lee McKnight and Joseph Bailey, editors, *Internet Economics*. MIT Press, 1997.

[21] Anirban Mahanti and Carey Williamson. Web proxy workload characterization. Technical report, Department of Computer Science, University of Saskatchewan, February 1999. `http://www.cs.usask.ca/faculty/carey/papers/workloadstudy.ps`.

[22] Andreu Mas-Colell, Michael D. Whinston, and Jerry R. Green. *Microeconomic Theory*. Oxford University Press, 1995. ISBN 0-19-507340-1.

[23] Lee McKnight and Joseph Bailey, editors. *Internet Economics*. MIT Press, 1997.

[24] Tracy Mullen and Jack Breese. Experiments in designing computational economies for mobile users. In *ACM Conference on Information and Computation Economies (ICE-98), Charleston, SC*, October 1998.

[25] Roger B. Myerson. Incentive compatibility and the bargaining problem. *Econometrica*, 47:61–73, 1979.

[26] Jenny Preece. *Human-Computer Interaction*. Addison-Wesley, 1994.

[27] Luigi Rizzo and Lorenzo Vicisano. Replacement policies for a proxy cache. Technical Report RN/98/13, University College London Department of Computer Science, 1998. `http://www.iet.unipi.it/~luigi/lrv98.ps.gz`.

[28] Alex Rousskov and Valery Soloviev. On performance of caching proxies. Technical report, NCAR, August 1998. `http://www.cs.ndsu.nodak.edu/~rousskov/research/cache/squid/profiling/papers/`.

[29] Scott Shenker. **???** In Brian Kahin and James Keller, editors, *Public Access to the Internet*. MIT Press, 1995.

[30] Hal Varian and Jeffrey K. MacKie-Mason. Generalized vickrey auctions. Technical report, Dept. of Economics, University of Michigan, July 1994.

[31] William Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[32] Stephen Williams, Marc Abrams, Charles R. Standridge, Ghaleb Abdulla, and Edward A. Fox. Removal policies in network caches for World-Wide Web documents. In *Proceedings of ACM SIGCOMM '96*, pages 293–305, 1996.