Biased Replacement Policies for Web Caches: Differential Quality-of-Service and Aggregate User Value*

Terence P. Kelly Yee Man Chan Sugih Jamin
{tpkelly,ymc,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

Abstract

Disk space in shared Web caches can be diverted to serve some system users at the expense of others. Cache hits reduce server loads, and if servers desire load reduction to different degrees, a replacement policy which prioritizes cache space across servers can provide differential quality-of-service (QoS). We present a simple generalization of least-frequently-used (LFU) replacement that is sensitive to varying levels of server valuation for cache hits. Through trace-driven simulation we show that under a particular assumption about server valuations our algorithm delivers a reasonable QoS relationship: higher byte hit rates for servers that value hits more. We furthermore adopt the economic perspective that value received by system users is a more appropriate performance metric than hit rate or byte hit rate, and demonstrate that our algorithm delivers higher "social welfare" (aggregate value to servers) than LRU or LFU.

1 Introduction

As World Wide Web usage has grown dramatically in recent years, so has the recognition that shared Web caches—L2 proxy caches serving corporate- or campus-sized LANs, and L3 caches embedded within wide-area networks—can play an important role in reducing server loads, client request latencies, and network traffic. Of the many Web cache replacement policies that have been proposed and evaluated [2, 6, 20, 27], none attempt to provide variable levels of service to clients and servers. This is surprising, because shared caches are obvious loci for differential quality-of-service (QoS) mechanisms. Disk space in shared caches is a strategically-placed resource, and it is reasonable to suppose that QoS differentiation might be achieved by devoting it to those who value it most.

It might be argued that because RAM and secondary storage densities are rising, and prices falling, the efficient management of shared caches via clever replacement policies is unnecessary. We take the opposite view. Computing power and prices have respectively risen and fallen for decades, yet scarcity remains a problem for nearly every identifiable resource, at least in some contexts. Experience shows that system user demand often grows faster than system capacity. Furthermore, it is widely observed that Web cache hit rates are proportional to the *logarithm* of cache size, and that replacement policies differ substantially in performance. A better replacement policy can therefore yield the same benefits as a *several-fold* increase in cache size. Given that cache space is a scarce resource, it is reasonable to apply *economic* principles and perspectives to the problem of allocating it.

We adopt the view that a networked resource ultimately is only as good as its users think it is. Therefore, we seek to design a feasible cache replacement policy that provides greater aggregate value to system users than do existing policies. This goal is known in the economics literature as *social welfare maximization*, and we think it is a better guide to design than conventional performance metrics such as hit rate and byte hit rate. To increase aggregate user value, we want a cache to deliver better service to users who value caching more. As a constraint on replacement policy design, we further require decentralized operation, in the sense that all

*Fourth International Web Caching Workshop, San Diego, California, 31 March–2 April 1999. Details available on the Web at `http://www.ircache.net/Cache/Workshop99/`.

$Id: wlfu.tex,v 1.29 1999/03/24 07:30:17 tpkelly Exp $

system components (clients, servers, and caches) pursue "self-interest," acting only upon local information. We make the last requirement not only to avoid communication overheads but also because we are interested in cache resources shared over a public internet, amongst autonomous, not generally cooperative system users.

We will do better at increasing aggregate user value if we can obtain meaningful measures of heterogeneous user values for service. In the work reported below we simply assume that users provide reasonably accurate estimates of the value they receive from caching, which are then used to bias a cache replacement algorithm. Since users are generally self-interested, they need appropriate incentives to provide accurate reports of their private valuations. One well-understood approach to providing incentives for value revelation is through a market or price system. In this paper we show that *if* reasonable valuation information is available, then it can be used to improve caching performance as measured by social welfare. We argue informally that if system users are furthermore charged fees related to their declared valuations, we expect that they will report reasonable values.

The use of price systems and market-like schemes for computer resource allocation has been proposed sporadically for at least three decades [22]. Markets are particularly appealing in distributed computing systems, because in some models they compute optimal resource allocations in a decentralized (and hence scalable) manner. One design method involves building "computational market economies" directly inspired by microeconomic theory. Examples of this approach include the Spawn distributed computing system [24] and Kurose & Simha's file allocation scheme [16]; see Wellman [26] for a review of "market-oriented programming" and its application to distributed resource allocation. Recently, an explicit "real estate" market in disk space has been successfully applied to the problem of Web caching [7]. An alternative design methodology, used in this paper, is to generalize a well-known algorithm in an economically meaningful way.

We describe a simple generalization of the familiar least-frequently-used (LFU) replacement policy that permits servers to increase the allocation of shared cache space to the URLs they host, thereby reducing server workloads. We show that if we assume a particular distribution of server valuations, our algorithm delivers substantially higher aggregate value to servers than ordinary LFU and LRU. Furthermore, high-value servers receive higher byte hit rates than low-value servers. Finally, our algorithm requires neither global information nor "altruism" in any system component, and can be retrofitted onto existing Web caches and protocols.

2 Biased Replacement Policies

We propose aggregate user value as the design objective for Web cache replacement policies. This objective contrasts with, for example, increasing byte hit rate. Byte hit rate is easily measurable and may seem objective. However, it ignores any heterogeneity in the valuation of caching services between system users, or by a given user over time. There is ample evidence that users do place heterogeneous value on network services, suggesting that it may be useful to bias resource allocation according to these differential values. For example, many dedicated private networks are built for users willing to pay more for lower latency. Residential users are demonstrating quite different willingness-to-pay for local access bandwidth (using analog, ISDN or xDSL phone lines, or coaxial cable connections). Therefore, we proceed from the view that even if it is not possible to perfectly observe all user values in order to maximize their sum, we should try to design a system that is likely to approximate this goal.

In this paper we focus almost exclusively on one class of system users: Web servers; in Section 4 we discuss a client-centered approach and some of the difficulties surrounding it. Servers experience reduced workloads (and their clients receive faster service) if their URLs are served from cache. To represent server valuations for caching, suppose that when a cache hit on URL u occurs, its origin server receives value equal to $W_u \times \mathrm{size}(u)$; equivalently, upon a miss the server incurs a cost of $W_u \times \mathrm{size}(u)$. We seek a replacement policy that attempts to maximize the sum over all cache hits of $W_u \times \mathrm{size}(u)$. In a more realistic model the cost of serving a request (and hence the value of not serving it) would include a constant term and would depend on a server's current load as well as the size of the requested object. Our main intent in this paper is to explore replacement policies sensitive to server valuations, so we have favored simplicity over realism in our choice of cost and benefit models. We plan to explore more sophisticated models in future work.

It is natural to describe cache replacement policies in terms of the sorting algorithms implicit in them [27]. Conceptually, a Web cache maintains a list of URLs sorted according to some key(s); a replacement policy removes URLs from the tail of this list. For example, least-recently-used (LRU) replacement sorts URLs on

```
for each request
    u ← requested URL
    if u is in cache
        deliver u to client
        record access time of u
        N_u ← N_u + 1
        [optional] charge (W_u × size(u)) dollars to server of u
    else
        retrieve u and W_u from server
        deliver u to client
        if size(u) ≤ cache size
            while (sum of sizes of cached URLs + size(u) > cache size)
                among cached URLs with minimal N × W, remove LRU item
            place u in cache
            record W_u and access time of u
            N_u ← 1
        end if
end for
```

Figure 1: The swLFU algorithm. N_u is the number of references to URL u since it entered the cache, and W_u is a weight supplied by the server associated with u indicating how much the server values cache hits; it may be "piggybacked" on HTTP reply headers and so adds little to server-cache communication overheads. Note that we must explicitly account for URLs larger than the cache, and that in our algorithm a URL is *always* guaranteed to be in cache after it is requested. Many authors fail to specify whether placement is mandatory or optional; Irani [14] and Cao & Irani [6] are among the rare exceptions.

time of last access, and LFU sorts on number of references. We define *server-weighted LFU* cache replacement (swLFU) as follows: For every cached URL u, let N_u be the number of references to u since it entered the cache[1] and let weight W_u describe the value per byte received by u's server when hits on u occur. The primary sort key in swLFU is $W_u \times N_u$, and the secondary sort key is time since last access. Note that if $W_u = 1$ for all u, swLFU reduces to ordinary LFU (with LRU as a "tie breaker"), and if $W_u = 0$ for all u it reduces to LRU. See Figure 1 for a pseudocode description of the algorithm. By sorting URLs on $N_u \times W_u$, swLFU retains those URLs that contribute most to aggregate user value per unit of cache space:

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

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

If servers must *pay* the cache for the value they receive (shown as an optional feature in Figure 1), we may say that swLFU attempts to *increase cache revenue*. The overhead of billing in a pay-for-service scheme need not be large; servers might pay caches at long intervals, e.g., monthly.

Weights in swLFU represent the extent to which servers value cache hits, and swLFU is sensitive to differences in server valuations. For simplicity, in this paper we say that servers directly reveal W_u to an swLFU cache. Is it plausible to assume that servers will reveal *truthful* weight values? In general no, but analyzing the extent to which servers would deviate from truth-telling in our scheme is beyond the scope of this paper.[2] It is reasonable to argue that if W_u are tied to payments, servers will be deterred from reporting inflated weights. Furthermore, provided that servers know they will receive more cache hits if and only if they declare higher weight, they have an incentive to report weights that reasonably approximate their true valuations.

[1] As Breslau et al. note [5], it is important to distinguish between Perfect LFU and In-Cache LFU. The former stores reference counts on all URLs, including those not in cache; the latter remembers N_u only for cached URLs. This paper discusses a variant of In-Cache LFU.

[2] The Revelation Principle of mechanism design (see, e.g., Mas-Colell et al. [18]) informs us that without loss of generality we can restrict ourselves to selecting an allocation mechanism—a market for buying higher sort keys—from those in which it *is* optimal to tell the truth. By restricting design to this set of *incentive compatible* mechanisms, we need not assess strategic lying. Thus, it is easier to obtain definitive results. Incentive compatibility analysis is an important facet of our ongoing work. See McAfee and McMillan [19] for a review of incentive properties in auctions and Varian [23] for a discussion of mechanism design as applied to "software agents."

```
if (200 != $htcode || ($method ne "GET" && $method ne "HEAD") ||
    $logtag eq "TCP_DENIED" || $logtag eq "TCP_NEGATIVE_HIT" ||
    $logtag eq "TCP_CLIENT_REFRESH" || $logtag =~ /^UDP_/ ||
    $logtag =~ /^ERR_/ ||
    ($url =~ m!^http:! &&
     $url =~ m!\.cgi/|\.cgi$|cgi-[bw]in|/cgi/|\?!i)) {
    $number_skipped++;
    next;
}
```

Figure 2: Perl code to filter NLANR access logs, used within loop that iterates over all requests. The regular expression that identifies dynamic content is similar to that used within the Squid cache. We reject requests with HTTP reply code other than 200 because we are interested in successful requests for data not present in browser caches. This code removed 38.1% of all requests at the PA cache site, 41.7% at the SV site, and 36.3% at UC.

	PA site	SV site	UC site
# servers	114,381	124,698	105,710
# URLs	3,412,105	3,744,274	2,884,598
# requests	7,011,622	7,897,659	5,568,112
Bytes req'd	131,665,275,664	161,620,444,331	127,346,723,989
Value req'd	264,025,996,908,451	319,466,493,484,667	264,106,487,028,562
Infinite cache			
size (bytes)	60,037,623,775	66,976,225,688	51,825,514,504
hit rate	51.3364	52.5901	48.1943
byte HR	54.4013	58.5596	59.3036
value HR	48.5422	57.4670	56.5745

Table 1: Summary statistics on our three request streams after filtering. Given our assignment of weights to URLs, "value requested" refers to the total value servers would receive if *every* request were served from cache, *including first requests for URLs*; infinite cache value hit rate refers to the fraction of value requested that would be delivered by a cache large enough to store all requested URLs.

3 Experiments

In order to evaluate the aggregate value and quality-of-service that swLFU delivers to servers, we conducted a number of trace-driven simulations comparing it with LRU and unweighted LFU (using time since last access as a secondary sort key). As input we selected three large request streams collected by National Laboratory for Applied Network Research (NLANR) caches at Palo Alto (PA), Silicon Valley (SV), and the University of Illinois at Urbana-Champaign (UC) during the period 15 August–28 August 1998 [12]. We filtered the raw NLANR access logs by removing all unsuccessful requests and requests for dynamic content; Figure 2 shows the actual Perl code used for this purpose. NLANR access logs record the number of bytes written to clients for each request rather than the size of URLs, and this field often varies across requests for the same URL (HTTP headers vary in size, URLs change, and clients sometimes abort transfers manually). We define the size of a URL to be the maximum recorded transfer size among all requests for it, and we consider URLs that differ only in a trailing slash to be identical. Table 1 displays summary statistics for our three request streams. We make no attempt to model cache consistency or "freshness" issues in any way; we assume that URLs remain unchanged for the duration of our simulation. We furthermore do not attempt to model aborted client-cache or cache-server connections. We wish to focus primarily on social welfare and QoS issues, so we have chosen the simplest possible evaluation framework.

3.1 Assigning Weights

In order to evaluate swLFU we must assign weights W_u to the URLs in request streams. Early experiments (not reported here) revealed that swLFU acts much like LFU in terms of its hit/miss behavior if URL weights are drawn from a narrow range, e.g., 1–10. The reason is the Zipf-like distribution of URL reference counts (see Figure 3). Because N_u values vary over five orders of magnitude, weights W_u must span a wide range if the replacement decisions made by swLFU are to differ much from those of ordinary LFU. For example, consider four URLs with reference counts of 1, 20, 400, and 8000. Observe that no assignment of weights in the range 1–10 to these URLs will alter their weighted sorted order as compared with their unweighted order. If servers are not sufficiently heterogeneous with respect to cache hit valuations, swLFU may offer relatively little advantage over LFU.

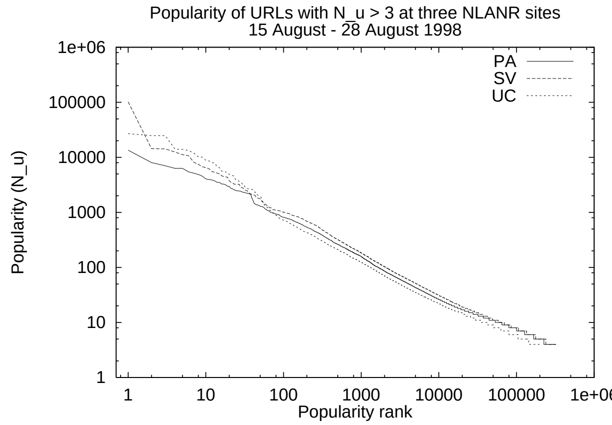

Figure 3: URL reference counts follow a Zipf-like distribution. Shown are reference counts N_u for all URLs in our three request streams with $N_u > 3$.

In our experiments we consider the case where servers *are* very heterogeneous. We do this by first assigning a unique integer identifier ID_s to each server, then assigning weights W_s to servers according to the formula

$$W_s = 10^{\mathrm{ID}_s \bmod 5}$$

and finally setting $W_u = W_s$ for each URL u hosted by server s (a server is simply the hostname or IP address component of a URL). The result is that W_u are drawn from the set $\{1, 10, 100, 1000, 10000\}$.

3.2 Results

We simulated LRU, LFU, and swLFU caches of sizes of 1, 4, 16, 64, 256, and 1024 megabytes for each of our three request streams (in all cases one gigabyte is less than 3% of infinite cache size). As measured by our primary performance metric of aggregate value delivered to servers, swLFU performs noticeably better than the other two algorithms. See Table 2 for all of our results and Figure 4 for plots of "value hit rate" (VHR) as a function of cache size. Value hit rate is a normalized measure of social welfare exactly analogous to byte hit rate (BHR). Whereas BHR is the number of bytes actually served from cache divided by the number of bytes requested, VHR is the value actually received by servers divided by the value they *would* receive if *every* request, including first requests, were served from cache. Note that in the special case of homogeneous valuations (i.e., all $W_u = k$ for some constant k), VHR is identical to BHR (just as BHR equals hit rate in the special case where all URLs have equal size).

		cache size (megabytes)					
		1	4	16	64	256	1,024
PA trace							
	LRU	1.24	3.54	9.09	15.09	20.32	24.95
VHR	LFU	1.53	3.89	8.73	13.36	16.09	20.62
	swLFU	2.78	6.50	13.93	18.56	22.74	31.55
	LRU	1.25	4.71	12.79	20.12	26.33	31.98
BHR	LFU	1.85	5.76	14.57	18.97	21.38	26.54
	swLFU	0.85	2.59	6.62	8.22	9.39	12.20
	LRU	1.88	3.59	6.17	9.45	13.11	18.37
HR	LFU	2.71	4.74	7.35	10.15	13.01	17.28
	swLFU	1.55	2.63	4.07	5.13	6.51	9.90
SV trace							
	LRU	3.53	9.07	14.75	22.94	29.09	35.00
VHR	LFU	3.99	11.74	16.55	20.11	25.25	28.84
	swLFU	7.02	14.13	19.51	27.06	33.33	39.14
	LRU	2.29	5.86	10.66	18.57	31.63	37.22
BHR	LFU	3.13	7.10	11.34	15.31	21.59	29.95
	swLFU	1.95	4.63	7.22	9.67	12.28	14.36
	LRU	3.15	5.40	8.58	12.10	16.01	21.57
HR	LFU	4.27	6.74	9.87	13.26	16.15	19.72
	swLFU	2.43	4.30	6.25	7.96	9.75	12.77
UC trace							
	LRU	2.59	7.27	15.07	23.51	33.13	39.83
VHR	LFU	3.52	8.45	15.04	22.36	29.58	34.66
	swLFU	4.93	11.45	21.11	30.06	36.58	43.81
	LRU	4.40	9.57	16.71	26.43	36.08	42.06
BHR	LFU	4.96	11.01	18.11	24.05	31.87	37.66
	swLFU	2.80	6.49	10.83	15.11	18.85	23.54
	LRU	4.26	7.44	10.73	13.79	17.18	22.20
HR	LFU	6.54	9.08	11.53	13.91	16.48	19.72
	swLFU	4.08	5.91	7.75	9.45	11.51	13.86

Table 2: Value hit rate (VHR), byte hit rate (BHR), and hit rate (HR) as function of cache size for LRU, LFU, and swLFU on three request streams. All numbers are percentages. Boxes emphasize that swLFU yields markedly higher value than LFU at a reasonable cache size.

Particularly striking is the difference in value to servers between unweighted LFU and swLFU: the latter delivers value hit rates between one third and one half higher than those of the former for large cache sizes (boxed data in Table 2). A value-aware algorithm can generate substantially higher social welfare than its value-insensitive cousin.

As expected, swLFU favors URLs with high weights. A positive correlation between service quality (byte hit rate) and weight value is evident when we examine the byte hit rates on URLs in each of our five weight categories (Figure 5). In a sense, our results are based on a sample size of one, i.e., one assignment of weights to URLs, and this accounts for the anomalous UC QoS curve, which contains a single outlier. The relationship between W_u and byte hit rate is as we would expect for the other two traces. As shown in the lower part of Figure 5, for the SV data a tenfold increase in W_u corresponds to a doubling in byte hit rate.

Of course, because swLFU does not consider all hits to be equal, it improves value hit rate at the expense of conventional performance metrics. For most cache sizes, LRU and LFU obtain hit rates and byte hit rates roughly twice as high as swLFU. We have argued that aggregate value to system users is a more appropriate performance metric than hit rate or byte hit rate, and our results show that swLFU provides both greater value and reasonable QoS differentiation to *servers*. A complete cost/benefit model, which is beyond the scope of this paper, would account for costs incurred and value received by *all* system users, including Web clients (who experience higher overall latency under swLFU) and network users (who encounter higher overall traffic due to swLFU's lower byte hit rate). Our results demonstrate that a familiar replacement policy can be modified to yield greater social welfare under a very simple and incomplete cost/benefit model. It is reasonable to suppose that value-sensitive algorithms can be designed to improve aggregate user value under more realistic assumptions, and this is the central goal of our ongoing work.

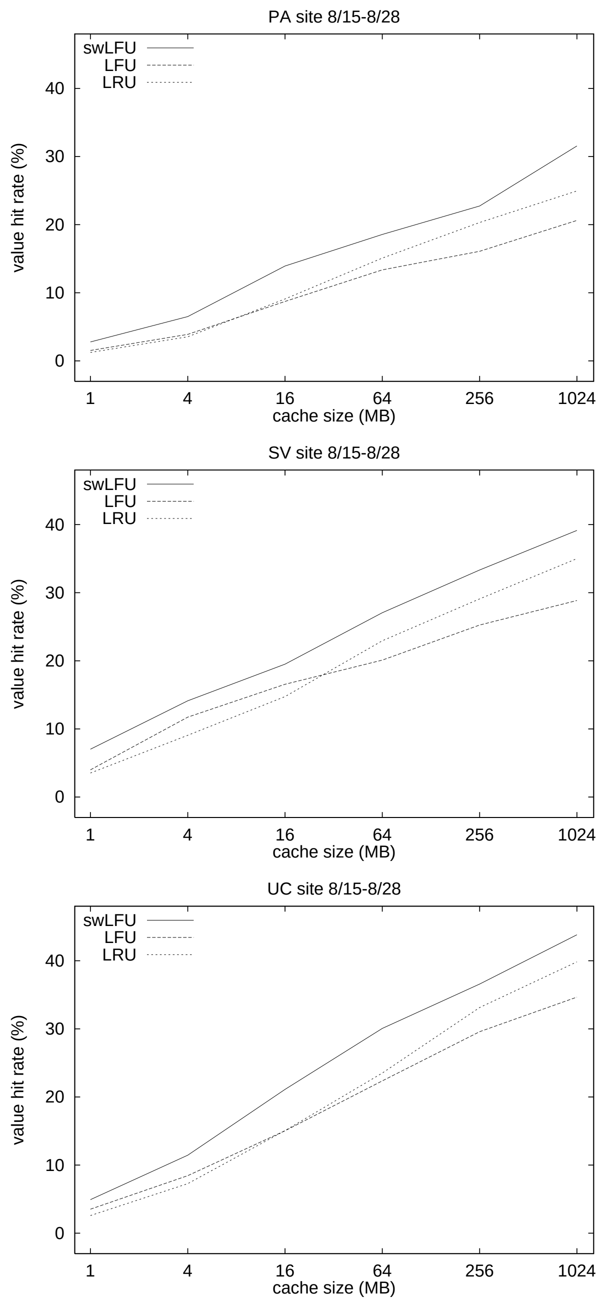

Figure 4: Value hit rate as function of cache size for LRU, LFU, and swLFU at three NLANR cache sites.

4 Client Bias

We might define a *client-weighted LFU* replacement policy (cwLFU) with primary sort key V_u given by

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

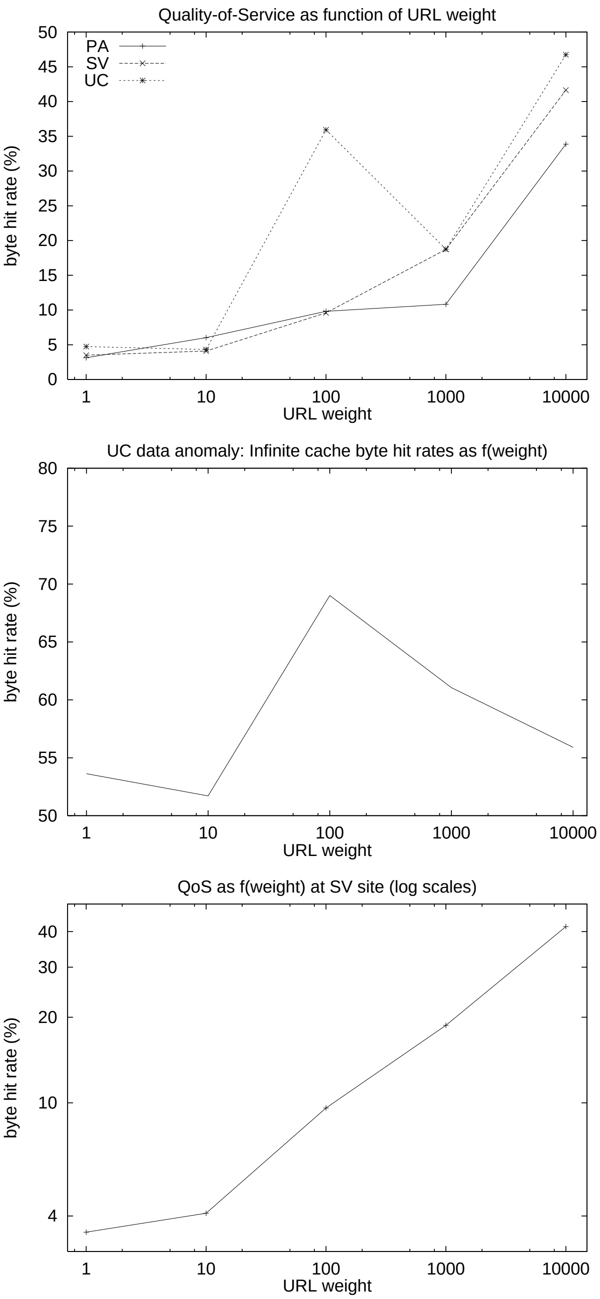

Figure 5: Quality-of-service as a function of W_u. *Upper figure:* byte hit rates for URLs in each of five W_u categories, 1024-MB cache simulations. The anomalous UC data point for $W_u = 100$ is purely the result of chance. *Middle figure:* infinite cache byte hit rates by weight category for UC site. The relationship between W_u and byte hit rate is unusual for the UC trace because *infinite cache* byte hit rates are remarkably high among URLs assigned weight 100. *Lower figure:* data for SV site only, logarithmic vertical axis.

where n_{iu} is the number of references to u by client i, w_i is the value client i receives per requested byte when a hit occurs on u, and therefore $w_i \times \text{size}(u)$ is client i's fee in a pay-for-hits implementation. This definition assumes that client value due to reduced latency is directly proportional to URL size; it is more reasonable to assume that latency will include a constant term representing connection setup time as well as a linear term representing transfer time. Again, at this stage we favor simplicity over realism in our cost/benefit models.

We do not present experimental results on cwLFU for two reasons. First, we rely on NLANR data for our empirical evaluations, and because of the way client IP addresses are anonymized in NLANR access logs it is impossible to track individual clients for more than one day, so trace-driven simulations of cwLFU for longer periods are impossible.

A second difficulty with cwLFU is more subtle. If we define the *mean weighted value* of a URL as $\overline{V}_u \equiv V_u/N_u$ where $N_u = \sum_i n_{iu}$, cwLFU's primary sort key is $\overline{V}_u \times N_u$, in other words, $\overline{V}_u$ plays the same role as W_u in swLFU. Clearly, if cwLFU is to differ significantly from unweighted LFU, the values of $\overline{V}_u$ must be different; conversely if $\overline{V}_u$ is similar for all URLs then cwLFU will cache roughly the same items as LFU. If client weights w_i are entirely independent of reference counts n_{iu}, the central limit theorem causes $\overline{V}_u$ to converge to the expected value of the distribution of the w_i. To illustrate this phenomenon, we obtained n_{iu} data from an NLANR access log, randomly assigned to clients integer weights w_i in the range 1–10, and computed $\overline{V}_u \equiv V_u/N_u$ for URLs with $N_u > 50$. As shown in Figure 6, values of $\overline{V}_u$ cluster strongly around 5.5. The net effect, confirmed by early experiments not reported here, is that by any performance metric cwLFU and ordinary LFU perform similarly.

It is reasonable to suppose that in the real world a correlation exists between the extent to which clients value faster service and the information they request; we might imagine that *Wall Street Journal* readers are wealthy and impatient, whereas the *National Enquirer*'s audience has less money and more patience. If this is true, then the "central limit theorem effect" might pose less of a problem for cwLFU. However plausible they may be, correlations between n_{ui} and w_i are purely speculative, and we do not attempt to model them in this paper. In future work we hope to devise cache management schemes sensitive to both client and server valuations; at this point we can only report that simple variants of weighted LFU do not appear promising as client-value-sensitive replacement policies.

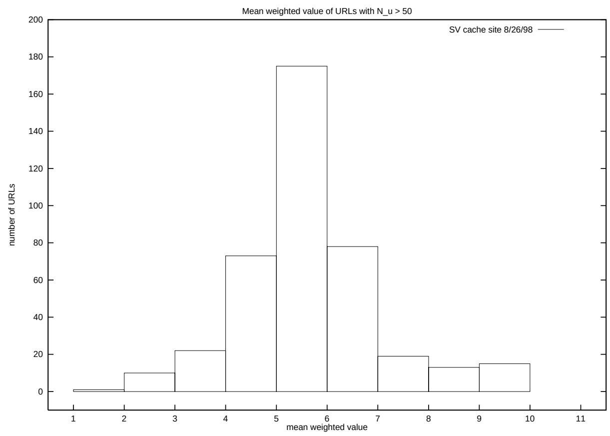

Figure 6: Histogram of mean weighted values $\overline{V}_u$ for popular URLs in NLANR's Silicon Valley L3 cache request log of 26 August 1998 (a busy day at a busy site) for a particular random assignment of w_i values to clients. Other assignments of w_i yield qualitatively similar results.

5 Conclusions & Future Work

We have shown that it is straightforward to generalize LFU to take into account heterogeneous server valuations of cache hits. Our simulation results demonstrate that under a particular artificial assignment of valuations to servers in actual trace data sets, swLFU delivers higher aggregate value to servers than LRU or LFU, and furthermore can provide reasonable variable QoS to servers.

In future work we shall explore more sophisticated assumptions about server behavior. Specifically, we conjecture that if servers adaptively adjust their reported hit valuations W_u in response to changes in workload, swLFU can provide a decentralized dynamic load balancing mechanism (see Karaul et al. [15] for a quasi-economic Web server load balancing scheme).

We are also investigating biased generalizations of other well-known replacement policies. Our results show that LRU obtains higher byte hit rates than LFU for large cache sizes, so it may be a more promising algorithm to generalize than LFU. Cao & Irani have proposed a "cost-aware" replacement policy closer in spirit to LRU than to LFU, GreedyDual-Size [6], and in future work we will include this algorithm in our comparisons.

6 Acknowledgments

Professor Michael Wellman provided valuable guidance to this project, and we thank Jonathan Womer for many useful discussions. Bill Walsh supplied valuable feedback on a late draft on very short notice, and David Ortiz offered a novel perspective on weighted LFU caching.

All authors gratefully acknowledge support from DARPA/ITO grant F30602-97-1-0228, from the Information Survivability Program. A RAND Corporation information survivability project led by Bob Anderson and co-sponsored by NSA, DARPA, and the Assistant Secretary of Defense for C^3I provided substantial early support. MacKie-Mason gratefully acknowledges support from an IBM University Partnership grant.

Trace data used in our simulations were collected by the National Laboratory for Applied Network Research under National Science Foundation grants NCR-9616602 and NCR-9521745.

Tom Hacker of the University of Michigan Center for Parallel Computing made available computing resources which greatly facilitated our experiments.

References

[1] Virgílio Almeida, Azer Bestavros, Mark Crovella, and Adriana de Oliveira. Characterizing reference locality in the WWW. Technical Report TR-96-11, Boston University Computer Science Department, 1996. `http://www.cs.bu.edu/techreports/`.

[2] Martin F. Arlitt and Carey L. Williamson. Internet Web servers: Workload characterization and performance implications. *IEEE/ACM Transactions on Networking*, 5(5):631–644, October 1997.

[3] Paul Barford, Azer Bestavros, Adam Bradley, and Mark Crovella. Changes in Web client access patterns: Characteristics and caching implications. Technical Report BUCS-TR-1998-023, Boston University Computer Science Department, November 1998. `http://www.cs.bu.edu/techreports/`.

[4] Paul Barford and Mark Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proceedings of the 1998 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 151–160, July 1998. `http://www.cs.bu.edu/faculty/crovella/paper-archive/sigm98-surge.ps`.

[5] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, and Scott Shenker. On the implications of Zipf's law for

Web caching. http://www.cs.wisc.edu/~cao/papers/.

[6] 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.

[7] 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 *Second Berlin Internet Economics Workshop*, May 1999.

[8] Ron Cocchi, Scott Shenker, Deborah Estrin, and Lixia Zhang. Pricing in computer networks: Motivation, formulation, and example. *IEEE/ACM Transactions on Networking*, 1(6):614–627, December 1993. A later draft, dated 15 May 1995, also exists.

[9] Mark E. Crovella and Azer Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *IEEE/ACM Transactions on Networking*, 5(6):835–846, December 1997. http://www.cs.bu.edu/faculty/crovella/paper-archive/self-sim/journal-version.ps.

[10] Carlos R. Cunha, Azer Bestavros, and Mark E. Crovella. Characteristics of WWW client-based traces. Technical Report BU-CS-95-010, Boston University Computer Science Department, July 1995. See Ref. [3] for a follow-up. http://www.cs.bu.edu/techreports/.

[11] Peter B. Danzig, Richard S. Hall, and Michael F. Schwartz. A case for caching file objects inside internetworks. Technical Report CU-CS-642-93, University of Colorado at Boulder Department of Computer Science, March 1993. http://catarina.usc.edu/danzig/ftp.sigcom93.ps.Z.

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

[13] Bernardo A. Huberman, Peter L. T. Pirolli, James E. Pitkow, and Rajan M. Lukose. Strong regularities in World Wide Web surfing. *Science*, 280:95–97, 3 April 1998.

[14] Sandy Irani. Page replacement with multi-size pages and applications to Web caching. In *29th ACM Symposium on the Theory of Computing*, pages 701–710, May 1997.

[15] Mehmet Karaul, Yannis A. Korilis, and Ariel Orda. A market-based architecture for management of geographically dispersed, replicated Web servers. In *Proceedings of the First International Conference on Information and Computation Economics (ICE-98)*, pages 158–165, October 1998.

[16] James F. Kurose and Rahul Simha. A microeconomic approach to optimal resource allocation in distributed computer systems. *IEEE Transactions on Computers*, 38(5):705–717, May 1989.

[17] Carlos Maltzahn, Kathy J. Richardson, and Dirk Grunwald. Performance issues of enterprise level Web proxies. In *Proceedings of the 1997 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 13–23, June 1997. http://www.cs.colorado.edu/homes/carlosm/public_html/mypapers.html.

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

[19] R. Preston McAfee and John McMillan. Auctions and bidding. *Journal of Economic Literature*, XXV:699–738, June 1987.

[20] 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.

[21] Sheldon Ross. *Stochastic Processes*. John Wiley & Sons, second edition, 1996. ISBN 0-471-12062-6.

[22] I. E. Sutherland. A futures market in computer time. *Communications of the ACM*, 11(6):449–451, June 1968.

[23] Hal R. Varian. Economic mechanism design for computerized agents. In *Proceedings of the First USENIX Conference on Electronic Commerce*, July 1995. http://www.sims.berkeley.edu/~hal/people/hal/papers.html.

[24] Carl A. Waldspurger, Tad Hogg, Bernardo A. Huberman, Jeffery O. Kephart, and W. Scott Stornetta. Spawn: A distributed computational economy. *IEEE Transactions on Software Engineering*, 18(2):103–117, February 1992.

[25] Carl A. Waldspurger and William E. Weihl. Lottery scheduling: Flexible and proportional-share resource management. In *Proceedings of the First Symposium on Operating System Design and Implementation*, November 1994.

[26] Michael P. Wellman. Market-oriented programming: Some early lessons. In S. Clearwater, editor, *Market-Based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1996. http://ai.eecs.umich.edu/people/wellman/Publications.html.

[27] 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.