The Case for Market-based Push Caching[*]

Yee Man Chan[†]
Member of Technical Staff
EnjoyWeb, Inc.
Santa Clara, CA 95054
ymc@enjoyweb.net

Jonathan Womer
School of Information
University of Michigan
Ann Arbor, MI 48109
jwomer@umich.edu

Sugih Jamin[‡]
Department of EECS
University of Michigan
Ann Arbor, MI 48109
jamin@eecs.umich.edu

Jeffrey K. Mackie-Mason[§]
School of Information
University of Michigan
Ann Arbor, MI 48109
jmm@umich.edu

Abstract

Web caches currently deployed on the Internet operate under a *pull* model in which client request streams determine the content of the cache. An alternative *push* model would allow web servers to pro-actively replicate their contents to caches. Given the finite amount of cache space, a question arises as to which objects should be kept in cache. In [6], the authors propose a push caching model whereby the content of the cache is determined by participating servers in a cooperative fashion. In this paper we explore an auction-based scheme that achieves an efficient allocation of disk space based on utilities revealed by both content servers and web caches. We show that our approach provides higher user valuation than traditional replacement policies without sacrificing overall hit rate. At the same time we solve the truthful revelation incentive problems associated with a cooperative approach. Our approach both implements e-commerce in caching service, and improves the infrastructure for supporting other e-commerce by providing quality of service differentiation.

Keywords: web caching, push-caching, incentive compatibility, differential quality of service

[*]This project is funded in part by DARPA/ITO grant F30602-97-1-0228 from the Information Survivability program.

[†]Yee Man Chan appreciates the equipment and financial support from EnjoyWeb, Inc.

[‡]Sugih Jamin is 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 gratefully acknowledges the support of an IBM University Partnership grant.

1 Introduction

The content of a web cache, as currently deployed on the Internet, depends on two factors: the request stream initiated by clients and the cache replacement policy. In this paper we describe a model whereby web servers may specify which of their contents they want cached. We call the former *pull caching* and the latter *push caching*. While pull caching has been successfully deployed, we think there will be instances in which push caching could be very useful. The main motivation behind push caching is that web servers may have information not available to either the clients or the web caches. A server that is about to release an object that is expected to be highly popular, for example, the trailer for an upcoming movie, an update to a popular software package, or an influential government report, may want to have the object pushed to caches on the Web before it announces the availability of the object. By doing push caching, not only will the server prevent itself from being swamped by requests, it will also decrease the download time experienced by its clients and, at the same time, reduce overall network bandwidth usage.[1]

Given the finite amount of space available at web caches, when cache space is fully utilized some existing objects must be evicted from the cache to accommodate a new object. Replacement policies for traditional pull caching rely on access history to make their decisions. The least recently used (LRU) policy evicts items based on their recency of use. The least frequently used (LFU) policy makes its decision based on each object's frequency of use. With push caching, the servers reveal private information to help cache manager to make object replacement decisions. This information might be based on access histories, but it might also

[1]"Push caching" is closely related to "mirroring". However, mirroring usually refers to a static or slow-changing replication of multiple objects or an entire site. Our notion of push caching allows a granularity at the level of a single object, and is highly dynamic so as to respond rapidly as network conditions and object value or desirability change.

incorporate other information available to the server but not part of the historical access log.[2]

In [6], the authors proposed a push caching scheme in which content servers cooperate with the web caches and with each other to determine the relative ranking of pushed objects. An object is not pushed to the web caches unless the demand for it exceeds a certain threshold. The value of this threshold and destinations to which objects are pushed are periodically tuned by a central administrator. The authors showed that geographic push caching can reduce network traffic by 26%.

In this paper we propose a market-based push caching mechanism in which cache space is auctioned off to servers. The advantages of using a market-based mechanism in push caching include:

1. The network (caching) resources can be allocated in order to maximize the aggregate value they provide to network users: *well-designed* markets yield *efficient* resource allocations.

2. A market mechanism decentralizes the cache space allocation decisions. With the distribution of allocation decisions, servers can employ more complex functions or large access histories in order to predict future popularity, or to calculate the value per request from putting specific objects into the cache. For example, LFU and LRU are very simple, not generally optimal forecasts of future requests.

3. A market mechanism also allows for access history to be kept at object servers and be communicated to the web caches in the form of value per request computed by the servers.

4. A server may choose to compute the bid value of an object independent of its access history or other indicators of expected popularity. For example, a server that wants to reduce the download time of its object may decide to bid high even if its client population is small. In general, the bidder can introduce information not available to a central cache manager.

In our earlier work [8] we proposed and investigated a market-based web caching system that provides differentiated quality of service, through basing the replacement policy on bids for the value of hits. In this paper we propose and investigate a market mechanism based on the value of *disk space* in the cache.

In [7] the authors propose a system in which cache spaces are distributed across the Internet. Web content servers can rent cache spaces to hold mirrors of their contents. Since this is a commercial product, no further details are available on the architecture of the system.

[2]For example, information about the expected frequency of use and value of latency for a new object, like a software update, that has no access history by definition.

There have been several pioneering efforts to apply market-based mechanism to the allocation of computing resources such as cpu cycles, disk space, etc. in both stand-alone and distributed systems [15, 11, 12, 3, 2, 16, 9]. Several researches assert that the use of market-based mechanism in allocation of computing resources has not seen wide-spread adoption because the limiting resources are sufficiently inexpensive that they can be over-provisioned. We believe that over-provisioning on the Internet will not be practical for at least three reasons:

1. By definition, resources on a network are shared resources. Unlike cpu or disk space for which a user may decide not to share with others, bandwidth and cache spaces on the Internet must be shared for network communication to happen. Given the highly variable usage pattern on the Internet—e.g., some users download large image files, others run video-conferencing, while still others only use the Internet to send email, to over-provision for the worst-case demand means low bandwidth users must subsidize the high bandwidth users.

2. While a user or an institution may over-provision resources for its own use, communication on the Internet goes through several administrative domains, any one of which may decide not to over-provision to carry someone else's traffic.

3. Network traffic has very high variance. The authors of [10] have shown that aggregate network traffic is long-range dependent, which means it has infinite variance. Over-provisioning for the worst-case demand will require a lot of resources, which will be highly underutilized most of the time.

For these reasons, we think that market-based mechanism can play an important role in push caching.

2 Web Model

We classify computers on the Web into three categories: All requests for Web pages are initiated by *browsers*. The permanent homes of Web objects are *servers*. *Caches* are machines on the Web that hold temporary copy of Web objects and offload browser requests from servers. Caches reduce the download latency experienced by browsers.

```
for each period
    move previously pushed objects to the space managed by LRU
    initialize available space to max cache size
    collect bids of <bidder, size, value per byte>
    sort the bids in descending order of value per byte
    for each bid in the sorted list
        if size is less than or equal to available space
            accept the bid
            decrement available space by size
            if the object in LRU space
                retrieve it
            else
                ask winning server to push object to cache
        else
            reject the bid
            if clearing price is not set
                set it to value per byte of this bid
        end if
    end for
end for
```

Figure 1: The uniform price auction algorithm. Notice that there is always a bid with value-per-byte equals to cache's reserve price.

```
if (200 != $htcode || ($method ne "GET" && $method ne "HEAD") ||
    $logtag eq "TCP_DENIED" || $logtag eq "TCP_NEGATIVE_HIT" ||
    $logtag eq "TCP_CLIENT_REFRESH" || $logtag =˜ /ˆUDP_/ ||
    $logtag =˜ /ˆERR_/ ||
    ($url =˜ m!ˆhttp:! &&
     $url =˜ m!\.cgi/|\.cgi$|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.

Periodically, caches hold a uniform price auction to determine which objects to accept for pushed caching. Before a new auction, the cache collects bids from servers. The bids are 3-tuples of the form <*server-id, size-of-object, value-per-byte*>. Bids are ranked in a descending order of value-per-byte. The n highest bidders that can fit in the available space are declared winners. If the cache has a non-zero reserve price, only bidders with value-per-byte higher than the reserve price are accommodated. The clearing price would be either the larger of the reservation price or the value-per-byte of the highest losing bid. The winning objects are then transferred by the cache from the servers. Figure 1 shows our auction algorithm.

Space not used to hold pushed objects is used to hold objects pulled by clients; this space is managed using the LRU replacement policy. In this study, we assume that cached objects are updated if the originals are modified. Our market-based mechanism can work with algorithms that ensure object freshness.

3 Experiments

In order to evaluate the aggregate value and quality-of-service that market-based push caching delivers to servers, we conducted a number of trace-driven simulations comparing it with LRU, unweighted LFU and Kelly et al.'s swLFU (for LFU and swLFU, we use time since last access to break ties). As input we selected three large request streams collected by the 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 [5]. 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.

3.1 Cache Parameters

Our cache has three parameters to be set by the cache owner: cache occupancy period length, cache size, and reserve price.

A period is the time interval between one auction and its successor. The contract between the cache and the servers states that the winning objects of the last auction can stay in the cache for one period. The authors of [6] set their period length to 30 minutes . In this paper, we use a period length of 20 minutes and cache sizes of 1, 4, 16, 64, 256 and 1024 megabytes (MB).

For all the results presented in Section 3.3, we set the reserve price to 0. In Section 3.4, we look into the effect of different reserve prices.

3.2 Server Bidding Algorithms

First, we assume all the servers we see in the trace are willing to push their objects to the cache and they will push every object that has a positive probability to be requested in the next period. (A server is simply the hostname or IP address component of a URL, we obtain it by the short Perl code segment shown in Figure 3). We further assume all servers use the same bidding algorithm in each simulation.

In our experiments we consider the case where servers are very heterogeneous in the value they place on having their objects cached (v_i). We do this by first assigning a unique integer identifier ID_i to each server, then assigning values v_i to servers according to the formula:

$$v_i = 10^{\mathrm{ID}_i \bmod 5}.$$

The result is that v_i are drawn from the set $\{1, 10, 100, 1000, 10000\}$.

The following subsections discuss the bidding algorithms we investigated. All algorithms are designed to forecast the number of future requests of an object in the next period. We assume that the product of this number, the object size, and the value-per-byte of the server would be the server's true valuation of having an object cached.

3.2.1 Regressed on Last Hour (RLH)

For each NLANR site, we first use the trace dated 1–14 August 1998 to compute the regression coefficients of each object. The regression coefficients are computed assuming linear model: $y = b_1 + b_2 x$, where x is the number of requests for the object in the last hour, and y is the number of requests for the same object in the next 20 minutes. Table 1 lists the computed regression coefficients.

To compute the value-per-byte for each object, a server first computes the likelihood that an object will be requested in the next 20 minutes from the regression coefficients (b_1, b_2) computed above and a trace of the last hour requests (xs) for its objects. The value-per-byte of each object is then obtained by multiplying

```
$url =~ s/\/$//;          # strip trailing slash
$url.= '/';               # add back the / for easy regex
$url =~ s!^[a-z]+://!!;   # remove ????:// at the beginning
$url =~ m!^([^:/]*)!;     # extract string up to ':' (followed by port #)
                          # or '/' (followed by path)
$server = $1;             # make it the server name
```

Figure 3: Server name extraction code segment.

NLANR Trace	b_1	std dev b_1	b_2	std dev b_2	R-squared
SV	-0.302478	0.01117111	0.303812	0.00269519	0.7764
UC	-0.293294	0.00988465	0.294441	0.00261484	0.6970
PA	-0.230651	0.01064222	0.238905	0.00574836	0.4131

Table 1: Regression Parameters for different NLANR traces.

$(b_1 + b_2x)$ of the object with the server's valuation of having its objects cached (v_i).

3.2.2 Perfect Foresight (PF)

We are also interested in knowing the performance of the system if all the servers know the exact number of requests in the coming 20 minutes (y). The value-per-byte will then be computed as v_iy. This should give us an upper bound on how well the system can perform, in that the servers are *actually* bidding according to their true valuations in this case; whereas in the other algorithms, they are only *estimating* their true valuations.

3.2.3 Limited Perfect Foresight (LPF)

Finally, we are interested in a performance upper bound on the class of algorithms that looks back one hour, for example, the RLH algorithm. Here we allow servers to look into the next 20 minutes but only submit bids for objects that also appeared in the last hour.

3.3 Results

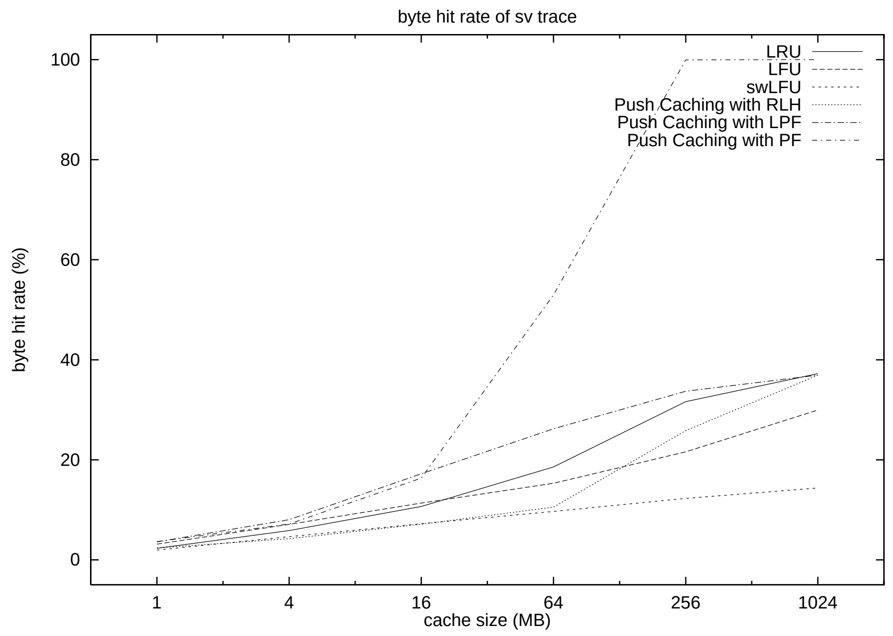
To evaluate whether our system delivers higher welfare than non-market based systems, we define a metric called value rate as:

$$\frac{\sum v_i h_i}{\sum v_i t_i}$$

Here v_i is as defined above, h_i the amount of data (in bytes) browsers obtained from the cache as opposed to from server i, and t_i is the total amount of data traffic (in bytes) seen at server i. We assume the benefit server i receives from caching is equal to the total benefits browsers using server i receive from cache, hence server i's utility function is $u_i(h_i) = v_i \times h_i$.

Figures 4, 5 and 6 shows that in terms of value rate, market-based push caching with all servers using the same RLH bidding algorithm does better than LRU and LFU from 1MB to 16MB. This is because the cache is congested. In other words, the total size of bids submitted to an auction is much bigger than the cache size and results in a high clearing price. The average total bid sizes per auction are 327.915MB, 246.636MB and 287.364MB for SV, UC, and PA sites respectively. The corresponding average clearing price are shown in Table 2.

We see that the swLFU algorithm, which is not incentive compatible , outperforms RLH for nearly all cache sizes. This is not surprising because we run periodic auctions, so our response to changes in request stream lags 20 minutes.



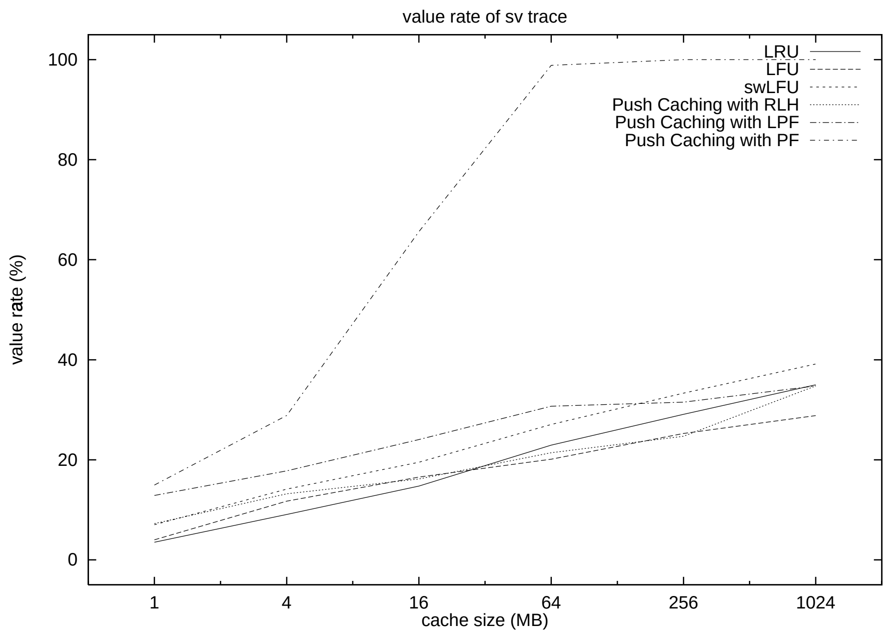
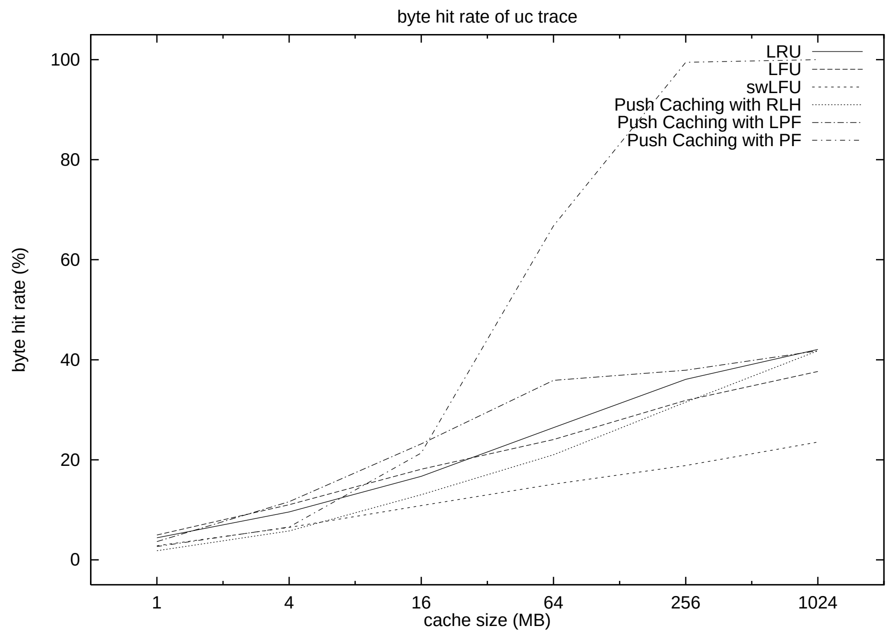


Figure 4: Byte hit rate and value hit rate as function of cache size for LRU, LFU, swLFU, Push Caching with RLH and Push Caching with LPF at SV NLANR cache sites.



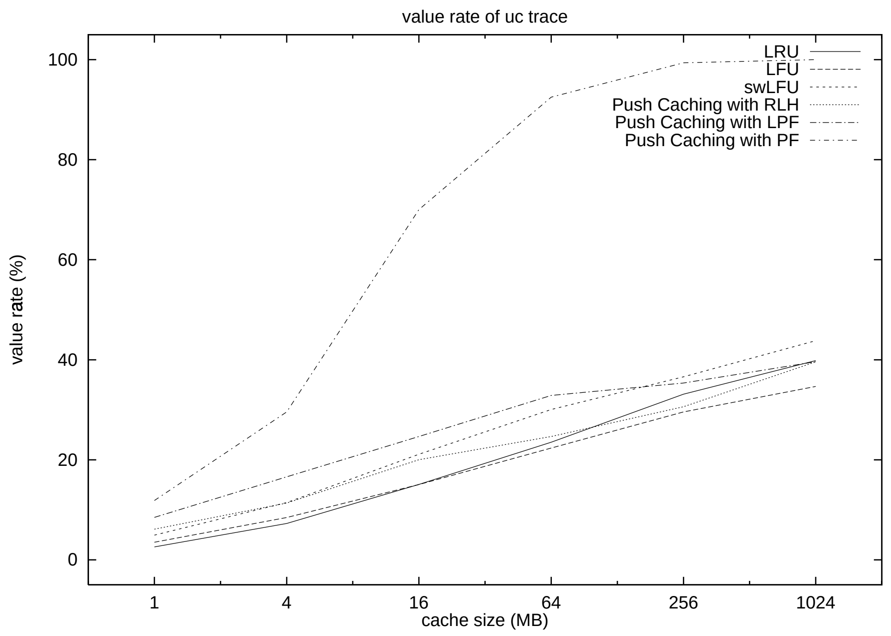
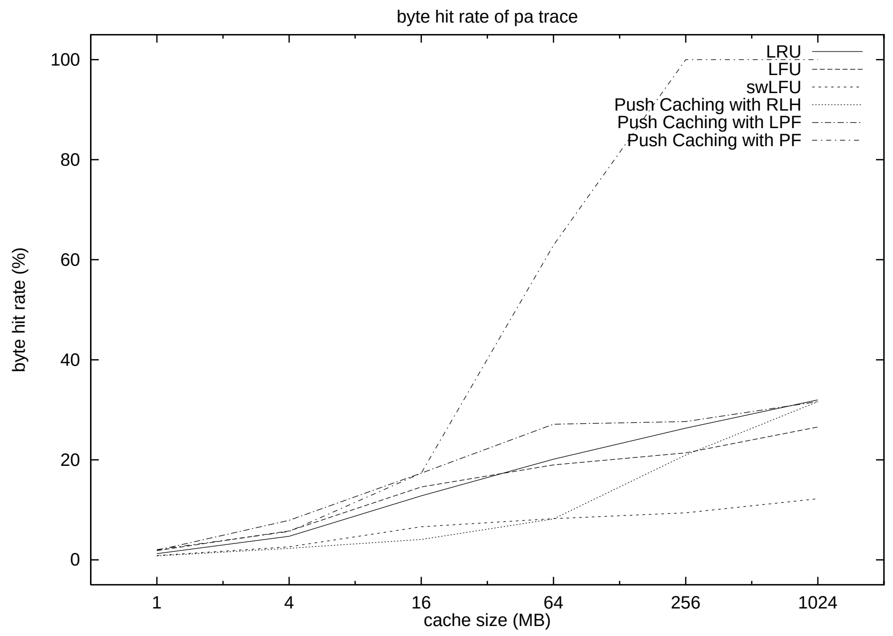


Figure 5: Byte hit rate and value hit rate as function of cache size for LRU, LFU, swLFU, Push Caching with RLH and Push Caching with LPF at UC NLANR cache sites.



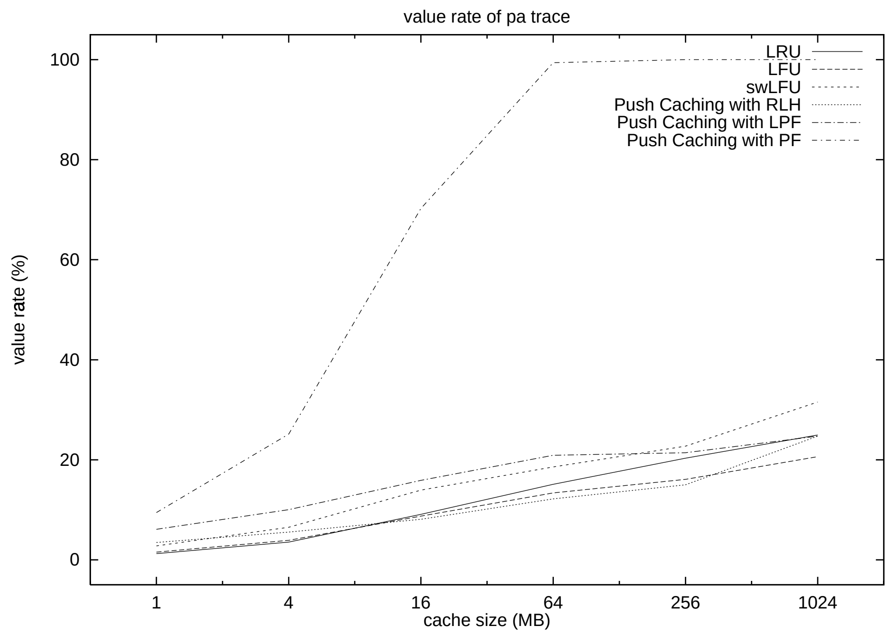


Figure 6: Byte hit rate and value hit rate as function of cache size for LRU, LFU, swLFU, Push Caching with RLH and Push Caching with LPF at PA NLANR cache sites.

With Perfect Foresight (PF) hit rate is limited only by the finite cache size; whereas with less than perfect foresight, the first an object is requested, it will result in a mandatory cache miss.

The performance of push caching converges to that of LRU when the cache size is large. This is correct because the rest of the space in the cache that is not filled up by winning objects is managed by LRU. Since the total bid size for each period under RLH algorithm is fixed, the increasing cache size leaves most of the cache managed by LRU. Hence, it is not surprising that our push caching system degenerates to a normal LRU-managed cache. This is a desirable feature because the cache will toggle between pull- and push-caching depending on demand.

3.4 Reserve Price Analysis

If the goal of the cache manager is to improve the welfare of the system instead of its revenue, it can set a non-zero reserve price. The effect of a non-zero reserve price is visible only when the offered load to the cache is near capacity. When the offered load is much higher than capacity and the cache is heavily congested, the clearing price could potentially already be much higher than the reserve price; on the other hand, if offered load is below capacity, the cache is not congested and all objects can be cached. Figure 3.4 shows that for a cache near capacity, some value of reserve price can improve both byte hit rate and value rate.

4 Future Work

We plan to implement our model on the Squid[4] web cache, using AuctionBot [13] to run the auctions. We envision that there are at least two possible ways to implement our system. The first one takes a de-centralized approach, for different flavors of web caches, patches are written to enable them to run auctions and understand the bidding protocols. In this case, servers would directly communicate with the web caches. In contrast, the second approach is centralized, there would be one auctioneer that represents a cluster of web caches. It would be responsible for deciding which objects should be replicated at a particular cache. Either approach has its own advantages and disadvantages. Further studies or real world experimentations are required to study the merit of different implementation approaches. One way to determine which model is the best would be to study the overhead generated by each approach.

Another territory of our future work is to come up with suggestions to help servers to construct their bids. To help our future attempt, we will consult the

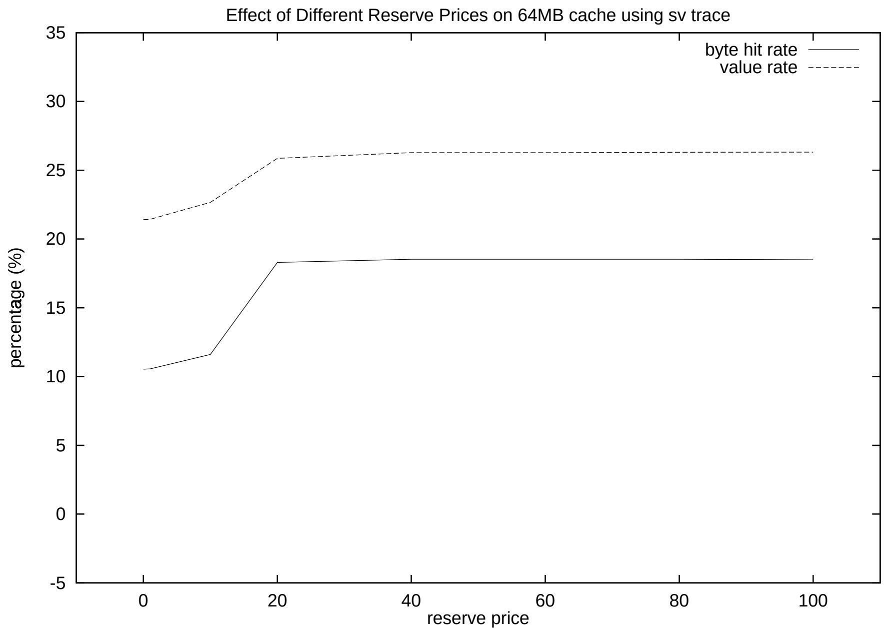


Figure 7: Effect of Different Reserve Prices on byte hit rate and value rate at 64MB SV site.

Trace cache size (MB)	1	4	16	64	256	1024
SV	16042.8	4030.45	732.972	9.50620	0.0469305	0
UC	11464.6	4030.45	442.811	36.1828	0.178155	0
PA	5515.89	2289.29	268.702	40.9102	0.110929	0

Table 2: Average clearing price for the auctions held in three NLANR cache sites.

work in the server-initiated replication literature [14, 1]. We believe the work done there would help us give advices to servers on the bid calculation - in our case, it would be the valuation of replicating an object in a web cache.

We also like to devise an efficient way to allow winning objects to authenticate to the appropriate web cache proxies.

Apart from security concerns, we also need a more realistic server value distribution than the 1, 10, 100, 1,000 and 10,000 we used. Another direction is to improve our current bidding algorithms. With more information and a more sophisticated prediction algorithm, we may be able to better approach the ideal case of perfect foresight.

Furthermore, we feel that a better way to model the reserve price of a cache would make it more realistic. For example, we can change our cache owner to a profit-maximizing agent instead of a welfare-maximizing one.

5 Conclusion

Motivated by the need to provide servers the autonomy to replicate objects, we developed a model that allows a proxy cache to accept pushed objects from servers. The push caching system is simple and highly flexible. While push caching requires bidding protocols, it remains transparent to the browsers. Hence no changes to existing protocols are required.

To tackle the incentive problem, we introduce a pricing mechanism for the cache space. The incentive compatibility of the mechanism ensures that servers are telling their true valuations. Since servers are bidding on their interest and their benefit is directly proportional to the frequency of future requests on the pushed objects, our system effectively delegates the computational overhead of managing caches to the servers. Our simulation results show that even with a very simple forecasting model, market-based push caching is able to deliver higher welfare than traditional cache management policies.

6 Acknowledgment

We thank Terence Kelly for many valuable discussions and the NLANR community, in particular Duane Wessels, for making the NLANR cache logs available and for answering our questions on Squid and its log format.

References

[1] Azer Bestavros. World Wide Web Traffic Reduction and Load Balancing Through Server-Based Caching. *IEEE Concurrency: Special Issue on Parallel and Distributed Technology*, 5(1):56–67, March 1997.

[2] David D. Clark. Adding service discrimination to the internet. Technical report, MIT Laboratory for Computer Science, Sept. 1995.

[3] 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.

[4] Squid Developers. Squid Internet Object Cache. http://squid.nlanr.net.

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

[6] James Gwertzman and Margo Seltzer. The Case for Geographical Push Caching. In *Proceedings of the Fifth Annual Workshop on Hot Operating Systems, Orcas Island, WA*, pages 51–55, May 1995.

[7] David Karger, Eric Lehman, Tom Leighton, Matthew Levine, Daniel Lewin, and Rina Panigrahy. Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web. In *Proceedings of the 1997 ACM Symposium on Theory Of Computation*, pages 654–663, 1997.

[8] Terence Kelly, Yee Man Chan, Sugih Jamin, and Jeffrey Mackie-Mason. Biased Replacement Policies for Web Caches: Differential Quality-of-Service and Welfare Maximization. In *Proceedings of the Fourth International Web Caching Workshop, San Diego, California*, April 1999.

[9] 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.

[10] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson. “On the Self-Similar Nature of Ethernet Traffic (Extended Version)”. *ACM/IEEE Transactions on Networking*, 2(1):1–15, Feb. 1994.

[11] Jeffrey K. MacKie-Mason and Hal Varian. Pricing the Internet. In Brian Kahin and James Keller, editors, *Public Access to the Internet*. Prentice-Hall, Englewood Cliffs, New Jersey, 1995.

[12] Jeffrey K. MacKie-Mason and Hal R. Varian. Pricing congestible network resources. *IEEE Journal of Selected Areas in Communications*, 13(7), 1995.

[13] University of Michigan AI Lab. Michigan internet auctionbot. http://auction.eecs.umich.edu.

[14] Renu Tewari, Michael Dahlin, Harrick Vin, and Jon Kay. Desgin Considerations for Distributed Caching on the Internet. In *Proceedings of the 19th IEEE International Conference on Distributed Computer Systems (ICDCS)*, pages 193–206, May 1997. Tech report version available on the Web at http://www.cs.utexas.edu/users/dahlin/papers/tr98-04.ps.

[15] 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.

[16] 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.