CACHE COHERENCY IN P2P COOPERATIVE PROXY CACHE SYSTEMS

Ankur B. Pal James Z. Wang Pradip K. Srimani*
Department of Computer Science
Clemson University
Clemson, SC 29634
{apal, jzwang, srimani}@cs.clemson.edu

ABSTRACT

Cache coherency plays an important role in the efficiency of a cooperative proxy cache system. In this paper we propose two new cache coherency policies, namely TTL-PI-C and TTL-PI-A to maintain the consistency of cached Web documents in our self-configured, self-managed P2P proxy cache system. Both policies take advantage of flowing Web cache lines among peer proxies that is a vital feature of our P2P proxy cache system. We exercise these policies in conjunction with our Criteria Weighted (CW) cache replacement algorithm and study the cache coherency effects using extensive simulations. We further extend the TTL-PI-A policy into Adaptive TTL-PI that uses the number of replicas known to a certain proxy to dynamically determine the TTL value. The performance study shows that the proposed policies improve the stale access ratios and the average cost bearing (in terms of bandwidth utilization) by commendable margins as compared to the traditional TTL based cache coherency policy.

KEY WORDS

Cooperative proxy cache, P2P, Web cache coherency.

1. Introduction

Cooperative proxy caching has been widely used in institutions to reduce network traffic and access latency. Many proxy servers inter-connected through the network, form a distributed proxy cache system, in which proxies serve requests from their clients as well as requests from their peer proxies. Our P2P cooperative proxy cache overcomes the limitations of existing cooperative proxy servers using an individual based model in which the aggregate effect of caching actions by individual proxy servers automatically distributes data closer to clients and balances the workload [1]. An important issue in proxy caching is cache coherency, i.e. the inconsistency between the cached Web document and the corresponding document in the original server. This inconsistent state can be witnessed more often due to dynamic rate of change in Web documents. Several cache coherency protocols are widely used to maintain conformance between the documents in the proxy servers and the corresponding document in the original Web server. In this paper, we propose new cache coherency mechanisms, which take advantage of the individual based model that provides the foundation for our P2P proxy cache system, to provide the combined benefits of strong [2] and weak cache consistency [3]. After briefly going through our proxy caching scheme in section 2, we discuss our proposed cache coherency policies in section 3. We then go on to describe the simulation method used to evaluate the proposed policies in section 4, and explain the cache coherency metrics in section 5, after which we analyze the results in section 6. Finally we have our conclusions in section 7.

** Srimani's work was supported by NSF Grant # ANI-0219485*

2. P2P Proxy Caching Scheme

In order to make this paper self-contained we briefly introduce our P2P proxy caching scheme in this section. Contrasting some other existing schemes in which the architecture of the proxy cache system is manually configured, proxies in our cache system automatically discover their neighbors and link themselves into a Virtual Proxy Graph (VPG) in which data search, data flow and data replication are implemented. Unlike traditional hierarchical cache, a Web document in a proxy cache system may be migrated to current proxy from its neighbors instead of the original Web server. To cope with the new data type and proxy cache architecture, we designed the Web cache line for our proxy cache system as depicted in Figure 1.

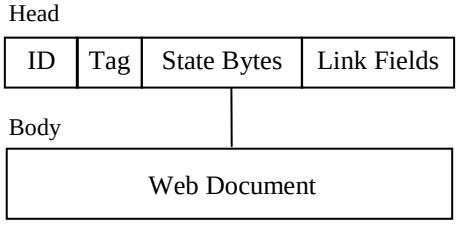

Figure 1: Web cache line.

The Web cache line is made up of two portions. The body portion is the Web document itself. The head portion consists of ID, Tag, state bytes and link fields. The ID field contains an UUID, which the URL of the cached Web document is hashed to. The Tag is the name of the document. State bytes are used to store information for cache coherency control and some other statistical

data, such as number of replicas of the document known to this node in the proxy cache system. A link field is organized as a pair of integer <NID, Dist>. NID is an integer used to lookup the neighbor table to get the neighbor's IP address. Dist is round trip distance in finding the cached Web document through that neighbor.

The heart of this proposed proxy cache system is the linkage of the Web documents across the proxies by flowing Web cache heads among the neighbor proxies. Because our P2P proxy cache system is designed on an individual-based model, the search algorithm runs on all proxies simultaneously and each proxy only queries its neighbors for Web documents. When a new request for a document comes in to a certain proxy from its clients, the proxy takes actions based on three different situations:

1. ***The entire Web cache line is cached:*** In this case, the Web document is sent to the client and the request is satisfied.
2. ***Only head of the Web cache line is cached:*** In this case, current proxy sends a request to the neighbor who has the shortest distance to a replica. The request is eventually propagated along the shortest distance route to the proxy having the replica.
3. ***Nothing is cached at this proxy:*** In this case, current proxy broadcasts a request to its neighbors and they in turn propagate the request to their neighbors until the request eventually reaches a proxy having a Web cache line that links to a replica. From this proxy, the request can be routed to the proxy having the replica, using the same process as in case 2.

In case there is no response until the query expiration time, the document is fetched from the Web server. Data search using this mechanism is different from simple flooding since the search uses link information from the Web cache line to find the document.

3. Proxy Cache Coherency

Cache coherency policies [2, 3, 4, 5, 6] basically fall into two categories; strong cache consistency and weak cache consistency. Both have two mutually inclusive goals:

1) To minimize the number of consistency checks between the proxy cache and the original Web server.
2) To reduce the number of stale accesses i.e. to reduce the condition that a document accessed from a proxy cache is different from that in the original server.

We could have chosen from a range of strong and weak consistency mechanisms to study their effects in our system. But note that the goals of both classes of mechanisms are mutually inclusive and dependent on each other. Reducing the frequency of consistency checks increases the likelihood of stale accesses and vice versa. An ideal cache coherency policy should limit the number of consistency checks to be no more than the number of updates in the Web servers, and concurrently strive to keep the number of stale accesses to zero. The existing cache coherency policies are far from being ideal. The coherency policies proposed in this paper strive to reduce the number of stale accesses, and at the same time keep the bandwidth utilization between the proxy and Web server low.

Our P2P scheme is based on peer co-operation where a peer is only aware of its neighbors. We propose cache coherency policies that can take advantage of the unique proxy caching scheme comprised of inherent simplicity and bound (connected) Web documents. The cached document has link fields that provide links to neighbors from whom the Web document is routed. This technique results in flowing Web cache lines across the co-operative proxies so that a Web document cached in one proxy can easily propagate a modification to all the proxies in the system having the same Web document. Hence a change in the state of a Web document can be propagated to a subset or the whole proxy system with minimal cost concerns. The cached documents also contain a field that preserves the number of replicas known to that proxy. This information suggests the global popularity of the cached document and can be used as a resource in devising cache coherency policies.

3.1 TTL-PI

This policy is based on the fact that traffic between cooperative proxy servers is more cost effective than between the proxy and original Web server. Also since our P2P cooperative proxy system reduces the amount of communication between the proxy servers as compared to multicast or broadcast techniques, we can afford to use the saved bandwidth to ensure the coherency of documents.

TTL-PI is a hybrid of strong and weak cache consistency policies. It uses the document TTL values to maintain weak coherence. Each individual cache node maintains a Time-To-Live (TTL) field in the cached Web document's URL. TTL is a priori estimate of how long the document will remain unchanged. This field is supported by the current HTTP protocol for cache consistency. Whenever there is a request for the cached Web document, the proxy checks if the TTL of the document is expired. If the TTL is expired, the proxy sends an "if-modified-since" HTTP request to the Web server. This special HTTP request contains the URL of the document and a timestamp. Upon receiving the request, the Web server checks whether the document has been modified since the timestamp. The Web server returns the status code "200" and the new data to the proxy cache if the data has been modified. Otherwise, a code "304" is returned.

Once a proxy server receives the latest document from the original server, other proxies are informed of the latest version by following the Web cache links. In other words the peer proxy invalidates the cached documents in other proxy servers taking advantage of the fact that it knows of a stale document. Hence instead of the Web server being

overloaded with the task of detecting and propagating invalidations as in case of strong cache consistency, the burden is delegated over to the proxy that the Web server is serving at that instant. This might seem a lot of work for a single proxy, but remember that our model is individual based and the proxy only has information about its neighbors. Hence invalidations would be sent only to a proxy's neighbors and those neighbors would then take care of propagating it to their neighbors and so on. In this way a single proxy does not encounter a large monolithic task of communicating a single invalidation message to all concerned proxies.

The Peer Invalidation (PI) can be implemented in two different modes:

TTL-PI-C: A conservative mode wherein the invalidating message will only be sent to the neighbors that current Web cache line has a link to so that only the stale replicas in a linked group will be invalidated.

TTL-PI-A: An aggressive mode wherein the invalidating message will be propagated to all proxy servers in the cache system so that all staled replicas can be invalidated.

The choice of mode influences the number of stale accesses and traffic between cooperative proxy servers. TTL-PI-C consumes less bandwidth amongst the cooperative proxy servers but averages to more number of stale accesses as compared to TTL-PI-A. However both modes reduce the network traffic as compared to strong coherency policies and also reduce the probability of stale accesses when compared to weak coherency policies. Hence both modes effectively make the two mutually inclusive goals mentioned in section 3 loosely coupled.

3.2 Adaptive TTL-PI

This policy is an enhancement of TTL-PI-A whose TTL value changes with the increase in popularity of the document. The state bytes of a Web cache line contain a field that holds the number of replicas of the Web document, an indication of the global popularity of that document. It represents the global cache information that is captured by each proxy node through data caching and message exchange. We take advantage of our ability to keep track of the number of replicas for a certain document through our caching protocol to design an adaptive mechanism that adjusts the TTL values dynamically. We denote this Adaptive TTL-PI as TTL-PI-N, where N means the number of replicas.

More number of replicas implies higher probabilities of stale accesses. This situation can be taken care of through the proxy invalidation suggested in TTL-PI. But the proxy invalidates the documents, only when the TTL expires. The TTL values are just an assumption that can be based on the size, origin or type of documents. To reduce the probability of stale accesses we reduce the TTL values dynamically for globally popular documents. This technique in effect, gives more preference for maintaining coherency to documents that are more popular in the system. This is done by periodically checking the number-of-replicas field in the Web cache line of a document. If the number of replicas goes above a certain threshold value then the TTL for that document is reduced.

Note that the threshold value is an absolute value whose lower bound is fixed at two, since a threshold value of one indicates that the cached document is only present in the local proxy cache and is not globally popular. A lower threshold value would invoke the TTL-PI-N policy more frequently than a TTL-PI-N policy with a large threshold value. This indicates that the performance of TTL-PI-N is inversely dependent on the threshold value, i.e. lower the threshold value better the performance of a TTL-PI-N policy.

4. Simulation Model

Our simulation model contains several components

1) The *VPG generator* constructs a Virtual Proxy Graph and is parameterized with the maximum distance to the neighbors and the maximum allowable links for each proxy. We apply a customized configurator to the set of proxies such that each proxy has a maximum of three neighboring proxies and a distance of three seconds to the Web server.

2) The *query sequence generator* creates query sequences for individual proxies. We used synthetic traces in our simulation and avoided web traffic logs since they may possibly reflect an inaccurate request ordering [7]. Web requests are known to follow a Zipf-like distribution [8]. The access frequency for each Web document *i* is determined as follows

$$f_i = \frac{1}{i^z \cdot \sum_{j=1}^{m} 1/j^z} \qquad (1)$$

where m is the number of Web documents in the system, and $0 \le z \le 1$ is the Zipf factor. A larger z value implies that some objects are accessed more frequently than others. When $z = 0$, the user access pattern is uniformly distributed, i.e. all objects have the same access frequency. Normally Web requests have a z value varying from 0.64 to 0.83 for different Web traces. Our simulation uses a z value of 0.75.

We use Poisson distribution [9] to simulate the inter-arrival time. If the average number of random occurrences per interval = m, the probability P of *a* occurrences in the interval is:

$$P(a) = \frac{e^{-m} m^a}{a!} \qquad (2)$$

The simulation assumes the interval to be one second and the average number of queries per interval to be six which may vary depending on network utilization by users and the number of cooperating proxies. The simulator also ensures that a proxy can encounter multiple queries in one clock tick, since a proxy can

serve many users at the same time. It is important to state that the term queries here suggests pure queries by clients and should not be mixed up with queries from neighboring proxies.

3) The *document modification module* is used to simulate the modification/revision of documents in the original server. To model it as close to the actual procedure we needed to know two things.

   a) The type of documents that belong to the two extremes, i.e. those that are most frequently modified and those that are barely modified.

   b) The frequency with which these types of documents are updated.

   We use the heuristic by Belloum et al [10] for approximating the modification model. It assumes that very large sized documents are not frequently modified. Moreover very small sized documents are almost static and are infrequently modified as well. Hence documents that do not belong to the two extremes have higher probabilities of modification i.e. those sized within the interval defined by the standard deviation are more likely to be modified. According to Web traffic analysis performed by Bestavros [11], the average document update probability is contained between 0.5% and 2.5%. We use an average value of 1.5% as default. The module is parameterized by the modification frequency and the probabilistic model of document selection.

4) The *cache replacement module* manages the cache space and enforces a replacement policy. We use our Criteria Weighted (CW) cache replacement algorithm which takes into account the access frequency, size, number of replicas, and time since the last access of the document. We calculate *V* using the function

$$V = \frac{\alpha}{f} + \beta \cdot s + \mu \cdot r + \nu \cdot t \qquad (3)$$

   where α, β, μ, ν are some predetermined constants We apply parametric values 1.0, 0.9, 0.3, and 0.6 to α, β, μ, and ν respectively [12].

5) The *coherency evaluator* keeps track of changing documents, fetches the new version of the cached document from the server when the TTL expires, updates the age of the document, and records the statistics related to coherency. We distinguished our documents into four types, namely http, gif, cgi-bin and other. We then parsed the IR cache traces [13] of 3 days to find the proportion of documents that belong to these types. The statistics obtained are as shown in Table 1. We apply the same distribution to our simulation and use the TTL values specified by Wooster et al [14]. For TTL-PI-N we decrement the TTL values by 1.5 hours and the minimum TTL value for any non-cgi document is assumed to be 1 hour. The threshold value for TTL-PI-N is set to 2, to get maximum performance.

Year '04	http	gif	cgi-bin	other
21st May	124612	69806	750	68
24th May	133507	65024	771	53
25th May	192140	112812	4154	25

Table 1: Statistics related to IR cache logs

5. Coherency Metrics

The main concern for cache coherency is whether the documents being accessed are the most recent version, i.e. how many stale documents forwarded to the users are different from the ones in Web server. Hence we measure the number of stale accesses in our simulation. This value should be low for reasonably good cache consistency mechanisms. We express this metric as the stale access ratio i.e. the ratio of the number of stale accesses to the total number of queries. The stale accesses in our system include the ones from neighboring caches as well. We also measure the average cost for each policy. Since we are using different cache coherency policies on the same replacement strategy and the same query sequence, the difference in cost can be determined from the required number of inconsistency downloads. For stale documents cached in the proxy server an inconsistency download refers to getting the updated document from the Web server. We do so by examining the responses of a coherency check query. A coherency check query is defined as a GET-If-Modified-Since request that is passed from a proxy cache to the origin Web server. We measure the average cost as the total number of instances where the Web server has to respond with a status code "200" and new data. Hence the lower the cost, lesser is the actual traffic between the proxy and the original Web servers. This metric should also have a low value for good policies.

A good policy should try to minimize the function

$$C = \alpha . S + \beta . R \qquad (4)$$

where *S* denotes the number of stale accesses and *R* denotes the number of GET-If-Modified-Since requests. α and β may vary depending on the network policies and preferences. If a network has low bandwidth-to-user ratio then we should have $\alpha < \beta$, whereas if the user tolerance for stale out-of-date documents is low then we should have $\beta < \alpha$. Our simulation assumes that $\alpha = \beta = 0.5$, i.e. user tolerance for stale documents bears the same weight as user tolerance for latency.

6. Simulation Results

We ran our simulator on DELL Precision 350 workstations equipped with 2.4 GHZ Pentium 4 processor and 1 GB DDR memory. For each set of parameters, a total of 400,000 queries were issued to the cooperative proxy simulator. We varied the cache space on individual

caching proxy from 2 MB to 22 MB while fixing the Zipf factor at 0.75.

6.1 Stale Access Ratio

Increasing cache sizes will allow more documents to be cached in the proxies thereby increasing the probability of stale accesses. The simulation results for stale access ratios for different coherence policies are reported in Figure 2. As expected the stale access ratio goes on increasing for increasing cache sizes. The original TTL mechanism has the worst stale access ratio amongst all contrary to significantly low stale access ratios for our proposed policies. For instance, TTL-PI-N shows a 77% improvement over the TTL policy in terms of stale access ratio for a cache size of 12 MB. This is due to the fact that we take advantage of co-operation between proxies and propagate the most recent and successful coherency checks amongst the same document in different proxies. Though this low stale access ratio comes with the cost of sending extra messages to neighbors, remember that it is much more efficient than a multicast or a broadcast to a specific/whole set of proxies.

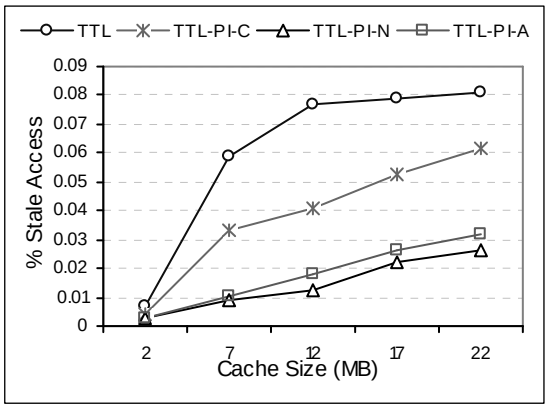

Figure 2: Stale Access Ratio for varying cache sizes

Also note that the gap between TTL-PI-A and TTL-PI-N increases with the cache sizes. This is because large caches act as a repository for more documents and hence increases the probability of more number of replicas, which brings the TTL-PI-N strategy into action more frequently. More documents with a considerable number of replicas result in more coherency checks and hence lower stale access ratios.

6.2 Average Cost

Figure 3 shows the average cost as a percentage of the total number of queries. Again the traditional TTL based policy is found out to be the costliest in terms of the number of downloads for content from the original Web server. For a cache size of 12 MB TTL-PI-N turns out to be 83% more efficient than the traditional TTL policy. This is a remarkable improvement considering that traffic between the proxy and Web server is a costly affair.

Also note that there is no significant difference in the cost of TTL-PI-A and TTL-PI-N. This infers that for almost the same cost TTL-PI-N achieves a higher degree of coherency than TTL-PI-A. Note that TTL-PI-N may incur more cost due to reduction of TTL's and more downloads from the Web server but it also reduces the number of stale accesses. Yet again there is a trade-off between the acceptable number of stale accesses and utilization of network bandwidth. It is worth to note that the cost for the original TTL policy keeps on increasing for larger cache sizes, whereas our proposed policies exhibit a relatively stable cost. Looking at Figure 2 and Figure 3 simultaneously, we can say that even though the traditional TTL based policy results in large number of downloads from the Web server, it fails to achieve the high degree of coherency accomplished by our proposed policies.

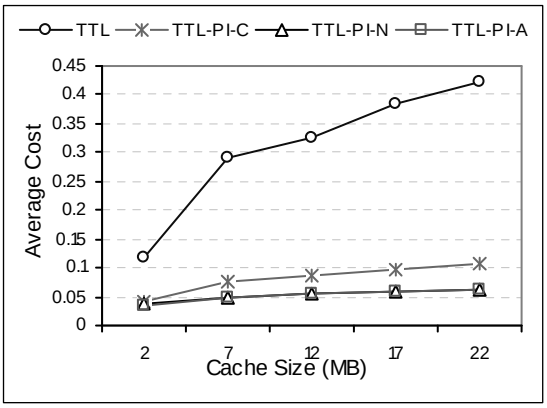

Figure 3: Average Cost for varying cache sizes

6.3 Futile Messages

We know that the performance enhancement of our policies is achieved through communication between the neighbors. So we measured the overhead introduced due to the extra communication. We specifically measured the average futile invalidation messages sent per query. An invalidation message to a proxy goes futile when the proxy already has an updated version of that document for which the message was sent. We include all such invalidation messages sent throughout the set of cooperative proxies, which implies that a futile invalidation message sent to a neighbor's neighbor and so on, are all included. We note that for the original TTL policy there are no such invalidation messages exchanged, only at the cost of increased stale accesses, and increased network communication between the proxy and Web server. Though Figure 4 is representative of the overhead, we can reduce this overhead by piggybacking the message content. Figure 4 depicts how TTL-PI-C saves lot of bandwidth among cooperative proxy servers. Futile messages for TTL-PI-C stay below 0.4 per query throughout the cache sizes, demonstrating greater efficiency with respect to the invalidation messages it initiates, as compared to the other two policies. However

when it comes to TTL-PI-A as well as TTL-PI-N, both almost have the same amount of futile messages which accounts for more bandwidth utilization between the cooperative proxy servers.

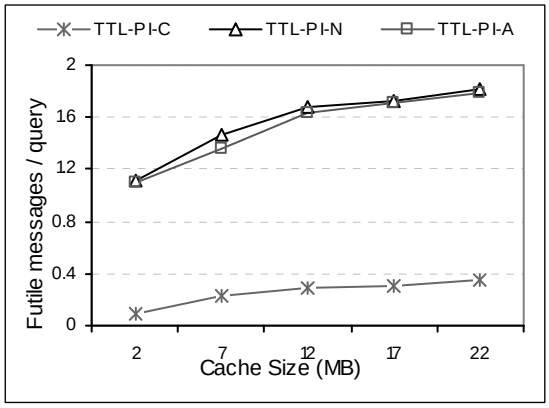

Figure 4: Futile messages per query

7. Conclusion

In this paper we study cache coherency policies for our P2P proxy cache system. We propose two new cache coherency policies (TTL-PI-C and TTL-PI-A) that utilize the flowing Web cache lines among the cooperative proxy servers to implement peer invalidation. Furthermore we refine the TTL-PI-A policy to take advantage of the fact that a Web cache line can keep information about its number of replicas known to this proxy and determine global popularity. We use simulation study to compare our coherence policies with the most widely used TTL based policy. Our performance study shows our proposed policies outperform the traditional TTL based policy by commendable margins in terms of stale access ratio and bandwidth utilization between the proxy and Web servers. Also TTL-PI-N gives the best performance when bandwidth concerns among the cooperative proxy servers are minimal, whereas TTL-PI-C is arguably better if bandwidth utilization among the cooperative proxy servers is costlier than stale accesses of Web documents.

References

[1] James Z. Wang and Rata K. Guha, Proxy Ecology – Cooperative Proxies with Artificial Life, *Proceedings of IEEE Conference on Intelligent Agent Technology*, Halifax, Canada, October 2003

[2] Chengjie Liu and Pei Cao, Maintaining strong cache consistency in the world-wide web. In *Proceedings of the 17th IEEE International Conference on Distributed Computing Systems*, May 1997.

[3] J. Gwertzman and M. Seltzer, World-wide web cache consistency. *International conference USENIX 1996*, San Diego, CA.

[4] Balachander Krishnamurthy and Craig E. Wills. Piggyback server invalidation for proxy cache coherency. In *Seventh InternationalWorld Wide Web Conference*,pages 185–193, Brisbane, Australia, April 1998.

[5] Balachander Krishnamurthy and Craig E. Wills. Proxy cache coherency and replacement – Towards a more complete picture. *In Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS)*, June 1999

[6] Vincent Cate. Alex -- A Global Filesystem. *In Proceedings of the USENIX File Systems Workshop*, pages 1-12, May 1992.

[7] Brian D. Davison, Web Traffic Logs: An Imperfect Resource for Evaluation, *Proceedings of the Ninth Annual Conference of the Internet Society (INET'99)*, San Jose, June 22-25, 1999

[8] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. *in proceedings of IEEE INFOCOM'99*, pp. 126-134, March 1999.

[9] B. Brewington and G. Cyeenko. How dynamic is the Web? *In World Wide Web* (May 2000).

[10] A.S.Z. Belloum and L. O. Hertzberger, Concurrent Evaluation of Web Cache Replacement and Coherence Strategies, *Transactions of The Society for Modeling and Simulation International*,Vol. 78 Issue 01, January 2002.

[11] A. Bestavros, Demand based document dissemination to reduce the traffic and balance load in distributed information system. *IEEE Symposium on Parallel and Distributed Processing 1995*, San Antonio, Tex., USA, pp 338-345.

[12] James Z. Wang, Ankur Pal and Pradip Srimani. New Efficient Replacement Strategies for P2P Cooperative Proxy Cache Systems. In *Proceedings of the 2004 Advanced Simulation Technologies Conference*, pages 97 – 103, Virginia, April 2004.

[13] IRCache Project. Originally sponsored by the National Science Foundation (grants NCR-9616602 and NCR-9521745), and the National Laboratory for Applied Network Research. http://www.ircache.net

[14] R. P. Wooster and M. Abrams. Proxy caching that estimates page load delays. *In Proceedings. Intl. WWW Conf*erence, Santa Clara, CA, Apr. 1997.