New Efficient Replacement Strategies for P2P Cooperative Proxy Cache Systems

James Z. Wang Ankur Pal Pradip K Srimani

Department of Computer Science
Clemson University
Clemson, SC 29634
{jzwang, apal, srimani}@cs.clemson.edu

Keywords

Cooperative proxy cache, P2P, Web caching, cache replacement, criteria weighted, victim pushing.

Abstract

The cache replacement algorithm plays a very important role in the overall effectiveness of a cooperative proxy cache system. In this paper, we study new cache replacement strategies in a self-configured, self-managed P2P proxy cache system. In this proposed P2P proxy cache system, cooperative proxies possess only local knowledge of the entire cache system and some limited global statistical information. Designing an efficient cache replacement algorithm (a global predicate) using only local knowledge is more challenging. We propose a Criteria Weighted cache replacement algorithm to manage the cache space in each individual proxy. The aggregate effect of cache replacement in individual proxies effectively optimizes the overall cache space in the entire cache system. Furthermore, we refine the proposed algorithm by pushing the replacement victim (the last replica known to current proxy) to one of its neighbors. We compare our protocols with some popular cache replacement algorithms using extensive simulations. The results show that the proposed CW and CW-VP cache replacement algorithms improve the cache hit ratio and average request latencies compared to the other algorithms.

1 INTRODUCTION

Proxy server is a special type of HTTP server used to allow users hiding behind a firewall to access the Internet. For security reasons, proxy servers have been widely used by institutions for serving the clients behind firewall. Although using caching proxy server can reduce both network traffic and document access latency, researchers have found that a single proxy cache can be a bottleneck due to its bandwidth and storage limitations [1,2]. The proxy server tends to be overloaded as the client number increases and hence causes a lot of cache missing. On the other hand, a single naive proxy cache system in a heterogeneous network environment might actually degrade the Web performance and introduce instability to the network [3]. To solve the problem, organizations normally use many proxies to serve as the cooperative proxy cache system.

Ideally, essential properties of a proxy cache system include *Minimized request latencies, Availability, Transparency, Efficiency, Flexibility, Robustness, Stability, Load balancing, Scalability, and Simplicity.* It is important to design and implement a Web caching scheme to possess all those properties. However, existing proxy caching schemes [1, 2, 4, 5, 6, 7, 8] do not satisfy all the requirements for such an ideal proxy cache system. Scalability and simplicity are the biggest problems in current Web caching schemes. To solve the problems, we proposed a novel P2P caching scheme [9] using an individual-based model in order to implement a self-configured, self-managed proxy cache system. In this proxy cache system, the cooperation among proxy servers is handled naturally by simulating an ecological system.

The most important component of any cache system is its cache replacement strategy. In this paper, we propose a network cache model that serves as the foundation of our cache replacement algorithm. After giving a brief introduction of the caching scheme in section 2, we propose, in section 3, the Criteria Weighted (CW) cache replacement algorithm based on the proposed network cache model and discuss its advantages over the other algorithms. We also discuss how Victim Pushing (VP) combined with CW cache replacement algorithm can reduce the user request latencies. In section 4, we design a simulation model to examine our observations and prove the effectiveness of the proposed CW cache replacement algorithm. The simulation results are presented in section 5. We have our conclusion and discuss future studies in section 6.

2 P2P PROXY CACHING SCHEME

In a distributed proxy cache system, proxies serve the requests from their clients as well as the requests from the peer proxies. They fetch the requested Web documents either from the Web servers or from the cooperative proxies to minimize the request latencies for their clients. In this paper, we use an individual-based model to design a self-configured, self-managed proxy

ecology in which individual proxy servers exchange data and information using some simple rules. The aggregate effect of caching actions by individual proxy servers automatically distributes the data closer to clients and also automatically balances the workload.

2.1 Virtual Proxy Graph

As an individual-based cache model, the proxy needs to find its neighbors with whom it exchanges data and information. Unlike some other existing schemes in which the architecture of the proxy cache system is manually configured, individual proxies in our cache system should automatically discover their neighbors and virtually link the cooperative proxies into a Virtual Proxy Graph (VPG). By forming the VPG, proxies are restricted to only exchange data and information with their neighbors. The neighbor proxies help each other in searching for cached Web documents and balancing the workload. The construction of the VPG follows some simple rules. First, neighbors must be close to each other. This implies that the cost of querying the neighbor for a requested Web document and fetching the document from the neighbor (if present) should be significantly less than fetching the document directly from the Web server. Also the overhead incurred by querying the neighbors for a document that is not present in the system should be minimal. Second, the number of allowable links to a proxy is determined by its available cache space, its network bandwidth, and its computing power. Normally a high power proxy server that has more cache space and larger network bandwidth will link with more neighbors. On the other hand, a proxy server that has limited network bandwidth should not link with many neighbors. Assigning the number of links to a proxy based on its computing resources is the key for automatic load balance.

There are several advantages of constructing a VPG. With the VPG, the proxies only exchange data and messages with their neighbors. So the management at individual proxy is simpler than those schemes that use multicast or broadcast to retrieve documents from other proxies. A VPG can be easily reconstructed based on the dynamic changing of workload and network environment. This feature makes the cache system not only adaptive to changes in network environment but also robust to the failure of some proxies. With the VPG, the aggregate effect of data movement among the proxy neighbors creates a life-like group behavior that automatically distributes the data closer to clients with less complexity. The VPG configuration process also provides a well-balanced distributed cache structure that does not depend on the underlying network architecture.

2.2 Network Cache Model

Cache was originally designed for hierarchical storage systems in computer architecture for fast access to frequently accessed data. Traditionally a cache line consists of cached data, tag and state bits. Tag field is used to identify the data page or instruction; state bits are used for cache coherency protocols. The size and content of the state bits vary depending on the cache coherency protocols. Figure 1 shows a traditional cache line.

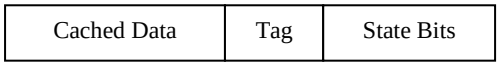

Figure 1: Traditional cache line.

Unlike in traditional hierarchical cache where cached data is normally a data page or an instruction from lower level storage devices, the cached data in Web caching is a Web document that may migrate to current proxy from its neighbors instead of being transferred directly from the original Web server. To cope with the new data type and proxy cache architecture, a Web cache line should include more information than that in traditional cache line. Figure 2 depicts a Web cache line in our proxy cache system.

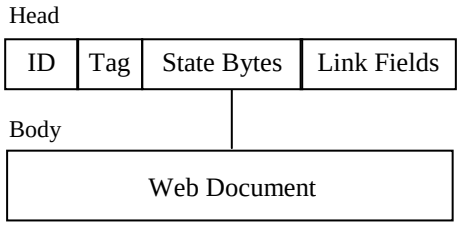

Figure 2: Web cache line.

A 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 as well as some other statistical data, such as number of replicas of the Web document known to this node in the proxy cache system. Because the cached Web document might be replicated from other nodes in the proxy cache system, link fields are needed in a Web cache line to provide the link information. 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 the neighbor specified by NID.

2.3 Data Search and Data Flow

The heart of this proposed proxy cache system is to link the Web documents across the proxies by flowing Web cache lines or Web cache line heads among the neighbor proxies. The data flow should bring the *Artificial Life* to the proxies and generate a *life-like* group behavior so that the aggregate effect of caching actions by individual

cache proxies automatically distributes data closer to clients interested in the cached data.

Efficiently searching the cached Web documents is the most important function in the proxy cache system. A good search algorithm should possess the following properties:

- ***Quickness:*** A search algorithm should quickly find the location of the cached Web document.
- ***Low cost:*** The overhead of the search should be low.
- ***Simplicity:*** The algorithm should be very simple to implement in individual proxies.
- ***Load balance:*** The search algorithm should not cause hot spots in the proxy cache system.

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 HTTP request for a document comes in to a certain proxy from its clients, the proxy takes actions based on three different situations as follows:

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 then its neighbors propagate the request to their neighbors until the request eventually reaches a proxy who has a cached 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 CACHE REPLACEMENT

Most existing cache replacement policies are proposed for a single proxy or a proxy cluster. Pitkow and Pecker [10] proposed a cache replacement algorithm based on dynamic user access patterns. A size weighted Web cache replace policy is discussed in [11]. Some LRU variants, such as LRU-MIN [12], LRU-Threshold [12], Size-Adjusted LRU [13] have also been discussed in past years. Kelly et al present a biased cache replacement algorithm [14] which is a simple generalization of LFU algorithm that is sensitive to varying levels of server valuation for cache hits.

Cache replacement policies have not been thoroughly studied in a cooperative proxy environment. Normally people simply adapt the cache replacement algorithms developed under the single proxy environment into the cooperative proxy cache system. However, simply adapting existing cache replacement schemes is not feasible in our individual-based cooperative proxy cache system. Unlike some other proxy cache systems where a directory server is aware of all cached documents in the cache system or the global cache status has been collected at local proxies through cache digest exchange, the individual proxy in our proxy cache system has only limited global information from its neighbors. Thus our cache replacement policy can only base on individual proxy's own local perception of the entire cache system. Another important feature of our proposed proxy cache system is that we replicate cached Web documents across the cooperative proxies based on demand in order to move the data closer to interested clients. The number of replicas of a Web document known to current proxy is another important factor in selecting replacement victim. This factor has never been considered in those existing cache replacement algorithms. So we have to design a new cache replacement algorithm particularly for our P2P proxy cache system.

Designing a cache replacement algorithm for a cooperative proxy cache system using local perspective of the global cache status seems not to be a good idea. However, there are many reasons to do so. First our P2P cooperative proxy cache system is based on an individual-based model. Each individual should maintain its autonomy and make decisions on their own perception of the system status. Second users in a cooperative proxy cache system normally share common interest and our data search and data flow have replicated the data across the network based on demand, thus local cache status can be a sample of global cache status. Third, the aggregate effect of cache replacement actions at individual cache nodes should create an effective global result based on our flocking-like individual-based caching model [9].

In our P2P cooperative proxy cache system, the network cache model divides the Web cache line into two portions. The head portion contains link information for cached Web document. So our cache replacement algorithms must treat the head and body of the Web cache line differently. Usually we should leave the head of a Web cache line in the cache system as long as possible to maintain the links among cached replicas. In case we have to remove the entire Web cache line, an invalidating message should be sent to its neighbors to

update the link information in the related Web cache lines. We discuss our cache replacement algorithms in subsequent subsections.

3.1 Criteria Weighted Algorithm

The criteria for choosing the replacement victim include access frequency of the document, size of the document, number of replicas of the document known to this proxy, and time since the last access to the document. Most existing proxy cache replacement algorithms consider only one factor in selecting a replacement victim. Since all those factors contribute to the efficiency of cache replacement, the Criteria Weighted (CW) algorithm for our P2P cooperative proxy cache system uses them all to pick a replacement victim. Access frequency and time since last access to the object are two local statistic parameters that reflect the popularity of the objects in current caching proxy. Access frequency may however build a high frequency count for a document that is subsequently never accessed again. The locality of time accesses for Web traffic often exhibit very different patterns and hence does not work well in proxy caches [15]. The number of replicas is stored in the state bytes of the Web cache line. It represents the global cache information that is captured by each proxy node through data caching and message exchange. Not only can the number of replicas reveal the global popularity of the object, but also it can be used to determine if certain object is the last resort in the cache system. A large number of replicas imply that a proxy will consistently find the object in its neighbor proxy if not in its own cache. The size of the Web document determines how much cache space it will take while being cached in the proxy. Because of the limitation in cache space, caching smaller Web documents should be the preference. However this may result in a situation where some small documents are never accessed once they are brought into the cache. Our approach is to blend all these concerning factors into a single function that determines the replacement victim when the cache is full. Assume access frequency, object size, number of replicas and time since last access to the object are *f*, *s*, *r* and *t* respectively for a cached Web document, the replacement value of the Web document *V* should be a function of these four factors: $V = F(f, s, r, t)$. We calculate *V* using the following simple function taken as the first approximation:

$$V = \frac{\alpha}{f} + \beta \cdot s + \mu \cdot r + \nu \cdot t \qquad (1)$$

where α, β, μ, ν are some predetermined constants. We calculate replacement value *V* for all cached Web documents and choose the document whose *V* value is largest as the replacement victim. The caching policy within an individual proxy is described as follows:

1) If the requested document is in the local cache of the proxy, then the document is served out of the cache and

a) *f* is incremented by 1.
b) *t* is updated to the current time.
c) the document is stored back in the cache.

2) If the proxy encounters a miss for the requested document in its local cache but finds the document in the neighboring proxy, then

a) *f* for the document is set to 1.
b) *t* is set to the current time.
c) *r* in the state bytes of the head of the Web cache line obtained from its immediate neighbor is incremented by 1. The link information is updated to point to the immediate neighbor from where the head of the document was fetched.
d) Check if there is enough space in this proxy to cache the Web document. If the space is not enough, calculate *V* for all documents in cache and evict the document with maximum *V*. Repeat this procedure until there is enough cache space to house the new document. Then add document to the local cache of the proxy. The proxy also needs to inform the neighbors to increment their *r* values.

3) If the proxy encounters a global miss and fetches the document from the Web server, then steps (a) through (d) of step 2 are repeated except that the Web cache head has the number of replicas in the state byte set to 1, and the link information is null.

3.2 Determination of parameters

Determination of parameters for our replacement function requires three steps

a) Finding the relative significance of each factor namely *f, s, r* and *t* and thereby determining the relation (order) between corresponding parameters.
b) Normalizing the contributing factors.
c) Deciding on an absolute value for each of the parameters.

Study by Chang et al [16] found that replacement policies based on size perform slightly better than those based on some other factors though it does not hold true in cases where the Web documents do not vary largely. Web documents however, have been shown to have a wide range of sizes, which generally vary from 1KB to 1MB. Hence β should make the most significant contribution of all parameters in our replacement value function. It is desirable to make β conspicuous, so that the algorithm would prefer replacing a single large size Web document with a large number of small size Web documents. The relative significance of each factor can

also be determined by the characteristic study of different Web traces. Some previous studies [16, 17] infer that Web traces follow a pattern where temporal locality effects are less dominant than specific global history for small cache sizes. That means the access frequencies of Web documents bear more weight than the time they are last accessed in cache replacement policies. So α is a more significant contributing parameter than ν in our replacement value function for small cache sizes. Moreover for large cache sizes replacement methodologies based on time since last access, outperforms algorithms based on frequency [18]. Hence for large cache sizes *v* is more significant than α. Analyzing the characteristics of artificial traces generated in our simulation model also concurred on the aforementioned observations. The number of replicas is the global popularity information. It is a controversial parameter in our P2P cooperative proxy cache system. Having a larger number of replicas in the cache system normally means the Web object is more popular and hence should not be replaced. However if we replace a Web object that has a larger number of replicas, the later requests can always retrieve a replica from nearby neighbor proxies. So the user response time will not be affected much although there is a cache-miss at local proxy. Based on those factors we assign the least weight to μ.

The determination of absolute values for the parameters α, β, μ, and ν has to be two-fold. Firstly we need to normalize the values contributed by each of the factors in *V*, namely frequency, size, number of replicas and time since last access. We then choose specific values for the parameters based on the facts mentioned above. To normalize the value contributed by the factors, we scale down the maximum contribution made by each factor in *V* to 1. This would require us to be aware of the minimum value of the number of accesses (*f)* and the maximum values of the size of documents (*s)*, the number of replicas (*r)*, and the time since last access (*t)*. These values can be theoretically determined. $f_{\min}$ is always 1 for a document that has been cached in a certain proxy. $t_{\max}$ theoretically depends on the TTL values of documents. $r_{\max}$ depends on the proxy structure and the resulting VPGs. $s_{\max}$ is dependent on the object set under consideration.

We now constrain the factors under normalization such that

$$f' = \frac{f}{f_{\min}},\ s' = \frac{s}{s_{\max}},\ r' = \frac{r}{r_{\max}},\ t' = \frac{t}{t_{\max}} \qquad (2)$$

Hence our replacement function *V* now looks as

$$V = \frac{\alpha}{f'} + \beta \cdot s' + \mu \cdot r' + \nu \cdot t' \qquad (3)$$

We then decide the scales of contribution of those factors in V. We keep the parameter values in the range {0, 1}. Our objective is to find an algorithm that uses all the four factors smartly to lead to an optimized algorithm. To avoid being overly biased to a specific algorithm we keep this range as small as possible. Hence we choose the range to be limited to 1. To be more elaborate if we had chosen a range of {0, 2}, then the contribution to *V* made by some parameter (say size factor β = 2) may always override the contribution to *V* made by another parameter (say time factor ν = 0.1).

After narrowing our choice for the absolute values of the parameters, we finally choose the values based on the facts discussed about the relative values such as

$\beta \succ \alpha > \nu$ (For small cache sizes) and

$\beta \succ \nu > \alpha$ (For large cache sizes)

where $\succ$ denotes slightly greater than and > denotes greater than.

We use these guidelines and constraints to select the constant values for α, β, μ, and ν in our performance study.

3.3 Victim Pushing

Traditionally cache replacement algorithms select an object as the victim to be deleted from the cache. However in our cooperative proxy cache system, when the number of replicas of a selected victim is one, deleting the Web document from proxy might not be a good idea. If we can push the Web document to a neighbor proxy, later reference to this Web document will not need to fetch it from the Web server, instead it can be retrieved from the neighbor proxy. Keeping the last replica in the cache system can also benefit other cooperative proxies. Victim Pushing (VP) is an attempt to sustain with the Web documents that will otherwise be removed from the cache system. This approach does not affect the hit ratio in local proxy cache but increases the global hit ratio. It also reduces the latency for future accesses to these documents, since the time to retrieve the Web document from the neighbor proxies is normally less than the time it takes to fetch the Web document from the Web server. Moreover Victim Pushing also reduces unnecessary duplication of documents in the cooperative caching system since a push operation is only considered when the neighbors of a certain proxy do not contain the document. Essentially Victim Pushing minimizes the penalty of deleting Web documents that are the only ones in the VPG known to a proxy. It also ensures that these are the only kind of documents that are sustained, thereby avoiding cache pollution.

There are two factors affecting the selection of the neighbor proxy to which the Web document will be pushed. One is the storage capacity of the neighbors and the other is the network distances from current proxy to the neighbor proxies. The best choice is to find out the neighbor proxy that has the maximum amount of cache space available and the lowest round trip time from the current proxy. Because our VPG configuration has ensured that only those proxies which have short network distances to current proxy can be the neighbor nodes in the VPG, we only consider the storage capacities of the neighbors in selecting the proxy to push the Web document. We choose the neighbor proxy that has the least storage utilization as the new repository of the victim object. An enhancement to this push strategy is to set a threshold for the neighbor proxies in accepting the victim object. We choose to push the victim only if there is a certain minimum amount of free cache space available in the neighbor proxy. This ensures that a candidate proxy has enough cache space to serve its own requests.

4 SIMULATION MODEL

Our simulation model consists of 25 proxy servers. Each proxy server is assumed to have identical computation power, storage capacity, and network bandwidth. Based on those assumptions, we used our automatic configuration process to configure those 25 proxies into a VPG with six connected uniform polygons, in which all proxies have either two or three neighbors.

The user request latency for a Web document is the time the user waits till the initial part of the Web document arrives at client's workstation. We assume the network distance from the client to its proxy server is 100ms. Hence when a certain Web document is found in the proxy cache, we can assume the user request latency to be 200ms. We also assume the round trip time between the proxy neighbors is 600ms and the average network distance from the clients to Web servers is 2000ms. Hence when a proxy or its cooperative proxies can not satisfy the client's request, the response time for the request should be 4000ms, ignoring processing and searching overheads. The assumption on the network distances are based on the data collected by running the benchmark Web Polygraph [19] on a Web caching appliance developed by Swell Technology [20]. We assume that the users have equal probability issuing Web requests at any proxies. We assume the average Web document size to be 60K. 35% of the Web documents have size less than 10K. 60% of them have size in the range from 10KB to 100KB. The sizes for the rest Web documents are in the range of 100KB to 1MB. The assumption on Web document size is based on the latest Web access statistics of several different Web servers [21,22].

The access frequency for each Web document i is determined as follows

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

where m is the number of the Web documents in the system, and $0 \le z \le 1$ is the Zipf factor. A larger z value corresponds to a more skew condition, i.e., some objects are accessed considerably more frequently than other objects. When $z = 0$, the user access pattern is uniform distribution, i.e., all the objects have the same access frequency. Normally Web requests have a Zipf-like distribution with z value varies from 0.64 to 0.83 in different Web traces.

In addition to our proposed cache replacement algorithms, we also implement LRU, LFU, and SIZE [17, 18] algorithms in our simulation model for comparison since we base our weighted algorithm on the factors that are predominant in these three algorithms. Comparing with these algorithms would provide an insight into the efficiency of our proposed cache replacement algorithms. As in most of the previous studies, we use hit ratio and average request latency as system performance metrics. To ensure all cache replacement algorithms are evaluated under the same data set, we generate a synthetic Web request sequence and apply the generated Web requests to every algorithm. We could have used Web traces [23] to evaluate the cache replacement algorithms. However, based on study conducted by Breslau et al. [24], Web requests follow zipf-like distribution with various zipf factors in different Web caches. We want to study how the Web request skew conditions affect the system performance. Using synthetic Web requests is easier to vary the zipf factors and hence is better for our evaluation.

Similar simulation results have been obtained by using different sets of parameters that satisfy our constraints discussed in section 3. Due to space limit, we only report the results obtained by applying parametric values 1.0, 0.9, 0.3, and 0.6 to α, β, μ, and ν respectively in our cache replacement algorithms.

5 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 200,000 queries were issued to the cooperative proxy simulator. Simulation warm up was taken care of since we started to collect statistic data only after the first 50,000 Web queries had been processed to avoid inaccurate statistic data due to startup of the simulation.

5.1 Performance under Different Cache Sizes

Increasing cache size will allow more Web documents to be cached in the proxies thus improving system performance. In this performance study, we vary the cache space on individual caching proxy from 2 MB to 22 MB while fixing the Zipf factor at 0.75. The simulation results are reported in Figure 3 and Figure 4.

As expected, the average request latency decreases when the cache size increases. When cache size increases, the proxies can store more Web objects in their caches and hence reduce user request latencies for those cached Web objects. The results in Figure 3 show that our proposed Criteria Weighted (CW) algorithm and its Victim Pushing enhancement perform the best in terms of average request latency.

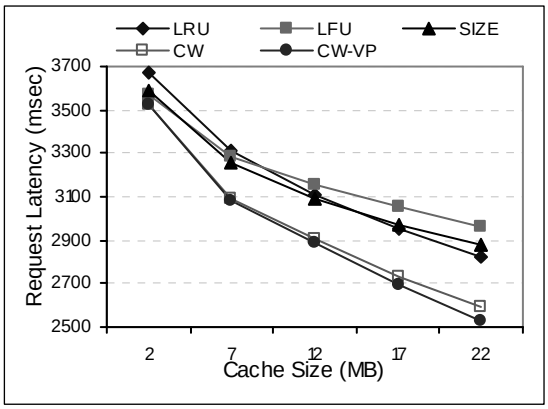

Figure 3: Average request latencies under various cache sizes.

The hit ratio of CW-VP as shown in Figure 4 is better in all cases. This is as expected since CW-VP is designed to sustain with the last replica of documents. This ensures that the documents are more uniformly distributed throughout the proxy cache system. The effect of CW-VP on cooperating proxies is that it maximizes the number of different documents that are stored in the system increasing the chances that a hit is encountered. This results in a larger hit rate for our scenario. Again the space between CW-VP and CW tends to increase with increasing cache sizes. This suggests that when the cache sizes increase CW-VP is more capable of distributing the documents uniformly within the overall cache space of cooperating proxies resulting in larger hit ratios. CW-VP shows a 56% improvement over LFU in terms of hit ratio when the cache size is 22 MB.

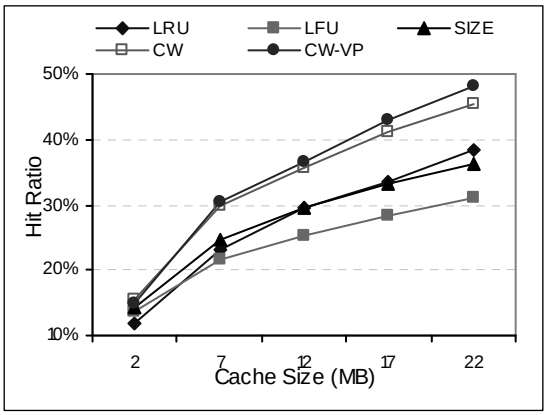

Figure 4: Cache hit ratios under various cache sizes.

5.2 Performance under Various Zipf Factors

Access skew condition determines the locality of the Web access. When data access skew factor increases, the average request latency decreases because increasing number of Web requests can be satisfied by fetching documents from the cooperative proxies due to data access locality. In this performance study, we maintain the cache space in each proxy at 15 MB. Since Web requests have a Zipf-like distribution with *z* value varying from 0.64 to 0.83 in different Web traces, we vary the Zipf factor from 0.6 to 0.9 in our simulation. The simulation results are depicted in Figure 5 and Figure 6.

Once again, our proposed algorithms outperform the other cache replacement algorithms. As shown in Figure 5, CW-VP algorithm yields the best performance in terms of average request latency. As for cache hit ratio, CW and CW-VP algorithms outperform the other algorithms by large margin in Figure 6. It is worth to note that when Web access skew condition is not severe, size becomes the dominant factor in cache replacement decision. On the other hand, when Web access skew condition is severe, the access frequency has more weight on the efficiency of cache replacement. Thus in Figure 5 and Figure 6, Size based algorithm outperforms LRU and LFU algorithm when Zipf factor is less than 0.75. On the other hand, LRU and LFU both perform better than Size algorithm when Zipf factor is greater than 0.75. Our proposed CW and CW-VP algorithms

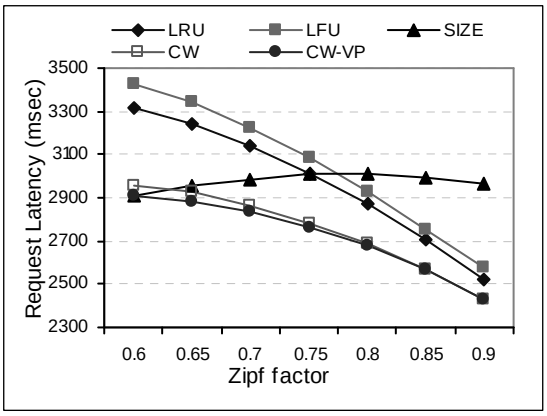

Figure 5: Average request latencies under different Zipf factors.

consistently outperform the other algorithms in any data skew conditions. This is because we carefully select the weight parameters for all the factors that affect cache replacement efficiency by analyzing the Web traces and experiments in previous studies. Our performance study has confirmed our analysis in section 3.1.1.

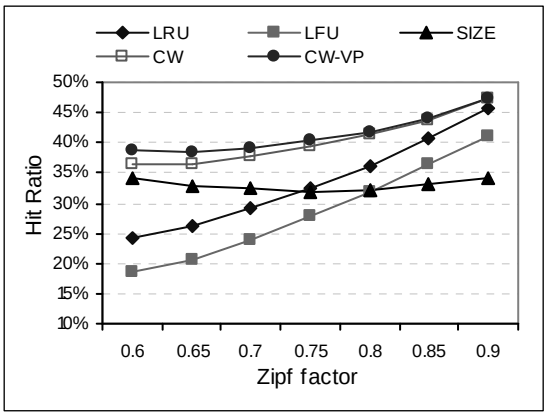

Figure 6: Cache hit ratios under different Zipf factors.

6 CONCLUSION AND FUTURE STUDIES

In this paper, we study new cache replacement strategies in a self-configured, self-managed P2P proxy cache system. In this proposed P2P proxy cache system, cooperative proxies possess only local knowledge of the entire cache system and some limited global statistical information. We propose a Criteria Weighted (CW) cache replacement algorithm that considers the combined effect of four criteria, i.e., access frequency, object size, number of replicas, and time since last access to the object. The aggregate effect of cache replacement in individual proxies effectively optimizes the overall cache space in the entire cache system. To take advantage of the cooperation among the neighbor proxies, we also design a Victim Pushing strategy to enhance the CW algorithm. We use simulation to compare our proposed algorithms with some popular cache replacement algorithms. Our performance study shows our proposed algorithms outperform the other algorithms by commendable margins.

As observed in our performance study, different factors weigh differently in various system conditions. In this paper, we determine those weight parameters by analyzing Web traces and some previous studies and then the same parameters are used by all proxies. In a heterogeneous network environment, system conditions such as Web access pattern and system resources vary in different proxies. Using different parameters for different individual proxies might be a better choice. We will study dynamically determining the parameters based on individual proxy's caching status. Further more, each individual proxy can obtain limited global information from its neighbors. We will exploit the neighbor cooperation in determining those parameters in our future study. Since cache coherency protocols influence the performance of a proxy cache system as well as cache replacement schemes, we are currently investigating the cache coherency problems in our P2P cooperative proxy cache system.

REFERENCES

[1] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel, A Hierarchical Internet object cache, *proceedings of USENIX Annual Technical Conference*, pp. 153-164, San Diego, CA, January, 1996.

[2] R. Malpani, J. Lorch and David Berger. Making World Wide Web Caching Servers Cooperate. *the Fouth International World Wide Web Conference*, Boston, Massachusetts, December 1995.

[3] A. Feldmann, R. Caceres, F. Douglis, G. Glass and M. Rabinovich, Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments, *Proceedings of InfoCom (1)*, pages 107-116, 1999.

[4] Duane Wessels, K Claffy, ICP and the Squid Web Cache, *IEEE Journal on Selected Areas in Communication*, 16(3):345-357, 1998.

[5] Scott Michel, et al., Adaptive Web Caching: Towards a New Caching Architecture, *the 3rd International WWW Caching Workshop*, Manchester, England, June 1998.

[6] Joe Touch, The LSAM Proxy Cache – a Multicast Distributed Virtual Cache, *the 3rd International WWW Caching Workshop*, Manchester, England, June 1998.

[7] V. Valloppillil and K. W. Ross, Cache array routing protocol v1.0, *Internet Draft, <draft-vinod-carp-v1-03.txt>*, February 1998.

[8] M. Rabinovich, J. Chase and S. Gadde, Not All Hits Are Created Equal: Cooperative Proxy Cache Over a Wide-Area Network, *Computer Networks and ISDN Systems*, 30(22-23):2253-2259, November 1998.

[9] James Z. Wang and Ratan K. Guha. Proxy Ecology - Cooperative Proxies with Artificial Life. *In Proceedings of IEEE/WIC IAT-2003*, Halifax, Canada, October 2003

[10] Jim Pitkow and Mimi Recker. A Simple Yet Robust Caching Algorithm Based on Dynamic Access Patterns. *the 2nd International World Wide Web Conference*, Chicago, Illinois, USA, October 1994

[11] S. Williams, M. Abrams, C.R. Standridge, G. Abdulla, and E.A. Fox. Removal Policies in Network Caches for World-Wide Web Documents. *In Proceedings of the ACM SIGCOMM '96 Conference*, August 1996.

[12] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching Proxies: Limitations and Potentials. In *Proceedings of 4th International World Wide Web Conference*, Boston, Massachusetts, USA, December 1995.

[13] C. C. Aggarwal, J. L. Wolf and P. S. Yu. Caching on the World Wide Web. *Knowledge and Data Engineering*, Volume 11, Number 1, Pages 95-107, 1999.

[14] T. P. Kelly, Y. M. Chan, S. Jamin and J. K. MacKie-Mason. Biased Replacement Policies for Web Caches: Differential Quality-of-Service and Aggregate User Value. *Proceedings of the 4th International Web Caching Workshop*, 1999.

[15] Ludmila Cherkasova. *Improving WWW Proxies Performance with Greedy-DualSize -Frequency Caching Policy*, HP Computer Systems Laboratory 1998

[16] Chung Yi Chang, Tony McGregor, Geoffrey Holmes. The LRU* WWW proxy cache document replacement algorithm, *Asia Pacific Web Conference 1999*, Hong kong, September 1999.

[17] Christoph Lindemann and Oliver P.Waldhorst, C. Lindemann and O. Waldhorst, Evaluating the Impact of Different Document Types on the Performance of Web Cache Replacement Schemes, *Proc. International Performance and Dependability Symposium (IPDS 2002)*, Washington, DC, June 2002.

[18] Pei Cao and Sandy Irani. Cost-Aware WWW Proxy Caching Algorithms. *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, 1997.

[19] Web PolyGraph, *http://www.web-polygraph.org/*.

[20] Tsunami: Web Caching Appliances, *http://www.swelltech.com/products/caching.html*

[21] Web server statistics, *http://www.lescroupiersrunningclub.org.uk/ace/logfile.php*

[22] Web Server Statistics for VSOHP, *http://satobs.org/ps.html*

[23] Web Traces and Logs, *http://www.web-caching.com/traces-logs.html*

[24] 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.