Adaptive Proxy Caching for Web Servers in Soft Real-Time Applications*

Albert M. K. Cheng Zhubin Zhang
Real-Time Systems Laboratory
Department of Computer Science
University of Houston
Houston, TX 77204-3010, USA
(cheng, zbzhang)@cs.uh.edu

Abstract

An adaptive cache proxy is developed to improve the performance of web access in soft real-time applications. It consists of client proxies and cooperative proxy servers with a server-side pushing schema. The large amount of heterogeneous data will be stored in the proxy servers and delivered to clients through computer networks to reduce the response time and network traffic. The adaptive proxy pre-fetches and replaces heterogeneous data dynamically in consideration of networks cost, data size, data change rate, etc. The simulation results show that the modified LUV algorithm has better performance in terms of hit rate, byte hit rate, and delay saving rate. With the cooperative proxy caching, it is shown that the performance of the proxy caching system is more predictable even if the proxies need to deal with a variety of data. The modified adaptive TTL algorithm has better performance in terms of the combination of temporal coherency and system overheads.

Keywords: adaptive proxy caching, web servers, soft real-time systems, mobile networking, temporal coherency.

1 Introduction

The World Wide Web (the "Web") has become a widely accepted channel for distributing a large variety of data and services in recently years. Although the Internet backbone capacity increases up to 60% per year, the scale of Web usage still far exceeds the capacity of the Internet infrastructure and leads to the relatively poor performance and low reliability of Web service. Therefore, it is challenging to deliver real-time data such as stock quotes, storm warning, and audio/video broadcasting to customers in a timely manner through the Web.

Researchers have been working on how to improve Web performance by means of proxy caching since the early 90's. Proxy caching is applied to store popular objects at the locations close to the customers, and has been proved to be an effective way to reduce the response time and network traffic, and to improve the robustness and scalability of the Web system. The following related work has been done to investigate different approaches to maximize the benefits of proxy caching.

1.1 Caching Architectures

The proxy servers can be organized to improve the performance of the caching proxy. A caching architecture can help caching proxies communicate and coordinate more efficiently to achieve a better performance. Caching architectures can be classified into hierarchical caching architecture, distributed caching architecture, hybrid caching architecture, and clustered caching architecture.

In hierarchical caching architecture [7], caching proxies exist in several network levels, such as bottom, institutional, regional, and national level. A hierarchical caching architecture has shorter connection times than distributed caching. Cache servers are added

* This work is supported in part by a grant from the Institute for Space Systems Operations. A 4-page preliminary version of this work has appeared as a work-in-progress paper in IEEE RTSS 2002.

to key access points to set up a hierarchy. This requires significant coordination among the participating caching servers. The extra delay may be introduced at every hierarchy level. High-level caches may become the bottleneck with long queuing delays. Copies of same documents at intermediate cache levels may introduce redundancy though it helps reduce latency.

In distributed caching architecture, there are no other intermediate cache levels other than the institutional caches, which serve each other and share the documents. Distributed caching systems have shorter transmission times than hierarchical caching systems [5]. It also allows better sharing and error tolerance. However, a large-scale caching system may have problems such as high connection times, higher bandwidth usage, and difficulties in system administration.

In hybrid caching, a certain number of caches cooperate at different levels of a caching hierarchy using distributed caching. The distribution at every network level can balance the workload of caching servers at different levels and reduce the overall retrieval latency. A cooperative proxy caching system [10] limits the cooperation among neighbor caches to avoid fetching documents from distant or slower caches, which could have been retrieved directly from Web servers at a lower cost.

Cluster-based Caching uses multicast groups for sharing cache objects, which is formed for a set of pages that are often accessed together. When a client wants to access one of these web pages, it can multicast the request to that group. Dynamic Web Caching [11] uses the "association rules" to find correlation that will be accessed based on another page that has been accessed. This approach can enhance the resource sharing among the group members and can increase hit rate. However, It introduces network traffic due to multicast query, especially when the group becomes large. In addition, distances among group members may not be close.

This paper will provide an approach using a distributed caching architecture with a default proxy server to achieve better scalability, efficiency and availability. This protocol avoids caching redundancy by saving URLs (or IP address and document path) uniquely among collaborative proxies by a hash function. Proxy servers cooperate with each other to make the most use of resources, as shown in Figure 1. The caching system consists of a client proxies, proxy servers, and web servers. Every mobile device works as a client and has its own Web browser and client proxy. The client proxy only maintains a part of cached objects because of the limited storage space of mobile devices. The other large amount of data will be saved in proxy servers. The proxy servers share cache objects among the group members. When the request of an object is sent to default proxy server from a client, the requested object will be checked with the cache objects in the default proxy server or group proxy servers through the hash table. If the requested object is found in the proxy group, the object is fetched from the proxy group and sent back to the client; if the object is not in the proxy group, the object is fetched from Web servers and sent back to the client. It can also reduce the connection overhead if the clients have persistent connections with the default proxy server.

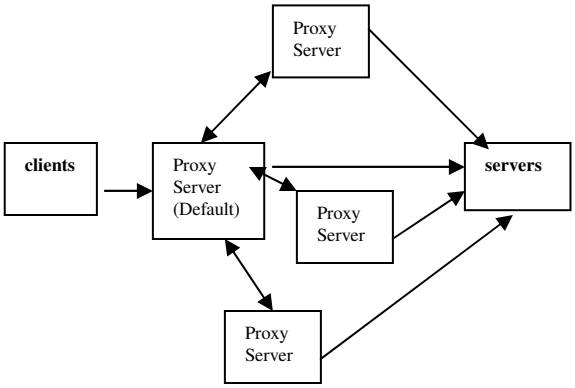

Figure 1: Proxy servers of soft real-time systems

Proxy groups can be clustered based on certain association rules as the cluster caching architecture has. The group can be formed according to the least delivery latencies among the proxy servers or based on the similar client access patterns. Also, the caching proxies can cooperate with other caches at the same level or at a higher level to form a larger scale system with more extensive resource sharing like hybrid caching.

The number of proxy servers should be controlled in a group to avoid the need to maintain a large hash table, and the delivery latencies among the cache proxy members should be limited. A large hash table may increase implementation overhead. If the distance between two member proxies is too far and the delivery latency is high, the proxy server may fetch the requested objects directly from web server.

Similarly, client proxies are organized into a client proxy group to share resource among the group members, as shown in Figure 2. There is a hash table to save the content information of its own proxy and other participating proxies. The limited storage spaces of clients are put together to increase the total cache size, which potentially leads to higher hit rate. First, the requested objects by the client browser are look up in its own client proxy or in the other group client proxies though the hash table. Second, the request will be sent to its default proxy servers.

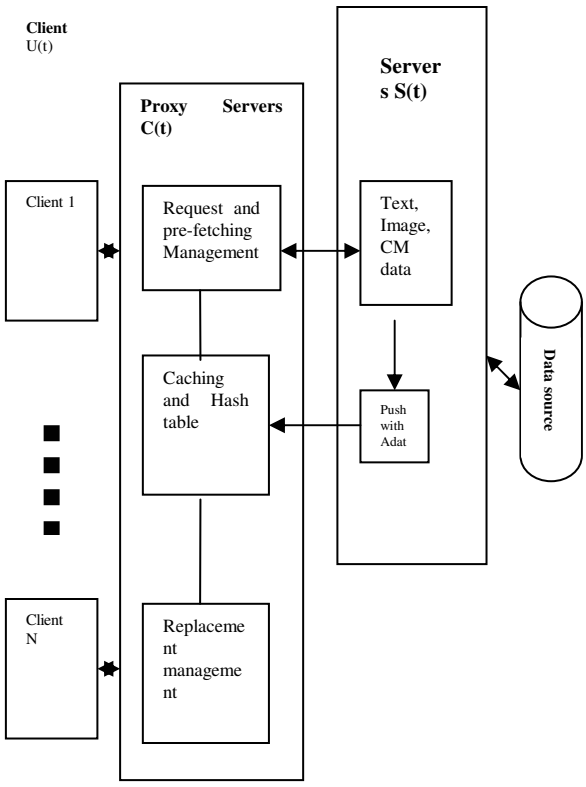

Figure 2: Configuration of proxy servers in soft real-time systems

A proxy server consists of the main parts such as a hash table of caching objects, pre-fetching management, and placement/replacement management, as shown in Figure 2.3. A hash table will be saved on the proxy servers to manage all the cache space in a server group. The requests of clients will be sent to default proxy servers. The hash table provides the exact caching location of a requested object. Each proxy keeps a summary of the locations of cached objects at participating servers.

Prefetching management predict the possible requests of clients based on the previous user access pattern, and get the possibly requested objects for clients between requests. Prefetching management also coordinate with cache coherency algorithm to fetch real-time data to keep data coherency and reduce latency.

1.2 Cache Replacement

A placement/replacement policy has a significant effect on the performance of proxy caching. In uniform caching environment, the size and the cost of each requested object are identical. A placement/replacement policy has significant effect on the performance of proxy caching. The previous works are divided into three categories:

a. Traditional replacement used such as Least Recently Used (LRU) and Least Frequently Used (LFU);

b. Key-based replacement such as LRU-MIN [7], LRU-Threshold [7], Lowest Latency First [9].

c. Cost-based replacement policy such as Lowest Relative Value [6], Least Unified Value (LUV) [3], Sized-Adjusted LRU [8], Hybrid [9], etc.

A new replacement policy is proposed and simulated to handle the heterogeneity of data types, to improve the hit rate and the byte hit rate, and to reduce the response time. The historic record, cost efficiency, and popularity of caching objects are reflected with adjustable weights in the current replacement policy based on Least Unified Value (LUV) [3]. In comparison with other replacement algorithms, LUV exhibits better performance in terms of hit rate, byte hit rate, and delay savings ratio in our trace simulation.

1.3 Prefetching and Data Coherency

Proxy servers can anticipate future objects and prefetch these objects into local cache to improve the hit rate and reduce the access latency. This approach predicts which cached objects a user might reference next and takes advantage of the idle time between user requests by either pushing cache objects to the user or pulling the documents from Web server [2].

Although the prefetching can increase hit rate and reduce delivery latency, the extra network traffic and the bandwidth wasting are introduced when the prefetched objects are idle. Every cache proxy must update its objects so that it can give users the objects or

data that are as current as possible. Time To Live (TTL) is used to decide if the data are invalid. As to real-time cache data, cache coherency becomes more important because the data are useful only if they are valid. Also, it is more challenging to maintain the coherency of real-time data whose change can be periodic or sporadic.

The adaptive TTL adjusts a cached object's TTL based on observations of its life-time. For example, if a cached file has not been modified for a long time, it tends to stay unchanged. Thus, the TTL attribute to a document is assigned to be a percentage of the document's life span, which is the current time minus the last modified time of the document. Studies [4] have shown that adaptive TTL can keep the probability of stale documents within reasonable bounds (5%). One the other hand, users cannot control the degree of staleness they are willing to tolerate. Users have to wait for expiration checks to occur even though they can tolerate the staleness of the requested objects.

In this paper, prefetching policy and adaptive TTL are applied to satisfy the requirements of Continuous Media (CM) and data coherency based on the available resources such as bandwidth and storage capacity. This policy can support the coherency of real-time data and different data types such as text, image, audio and video. Also, this policy can provide the function to let user specify the degree of staleness and the coherency predicable performance for soft real-time applications.

1.4 Multi-media caching

A lot of work has been done to improve the scalability, the latency, the bandwidth utility, and the stream quality in multi-media applications. For example, A Multimedia Proxy Caching (MCaching) was proposed to complement the end-to-end architecture for the delivery of quality adaptation, layered-encoded streams over the Internet [12].

1.5 Objectives

Based on previous related work on proxy caching, this study is to propose an adaptive proxy caching scheme for Web servers in soft real-time applications, and to improve the quality of the caching system in terms of access time, load balance, data coherency, simplicity and ability to deal with data heterogeneity. The specific objectives of the investigation are to improve, by means of simulation analysis:

Replacement algorithm to handle data heterogeneity in terms of simplicity, hit rate, byte hit rate, delay saving, etc.

Adaptive TTL to maintain the cache coherency with relatively low network traffic.

2 Algorithms

2.1 Hash Table Algorithm

A hash table is maintained to store the exact location of objects and map the Uniform Resource Locator (URL) to a hash key, which helps retrieve the exact location of objects or remove the objects from the table according to a replacement policy. URL is used to identify a Web document and contains the machine name or IP address, and the file name. These requests from clients will be checked with the hashing table for potential hits before the requests are sent out. Proxy servers check the availability of a request object before the request is forwarded to destination servers. There are three cases to handle.

(1) The location of object is in the hash table and is on the default servers. Just fetch it from the default proxy server.

(2) The location of object is in the hash table and is on the other proxy servers. Fetch the requested object from the proxy server where the object is stored and send the object to clients.

(3) The location of object is not in the hash table. It means that the object is not saved in the proxy server group. The request has to be sent to Web servers to get the object from the data source.

In this study, the reference point of IP address and documents are saved in the harsh table to reduce the latencies related with Domain Name Service (DNS) lookup. The reference saving can avoid the times for a DNS lookup and the persistent connection with a default server can save the connection times.

2.2.2 Replacement Algorithm

Most caching proxies handle text, images and continuous Media (CM) by different policies and priorities since the size of CM is much larger and is real-time data. In this approach, the text, image and CM are treated by one replacement policy. CM will be allocated with fixed and relatively larger caching space to avoid exhausting caching space and bandwidth by large CM data. The remainder of CM data will be fetched from the servers when they are requested by clients. The replacement policy can also be simplified and be more adaptive.

The replacement policy uses the Modified LUV (Least unified value) that is found to be effective in improving the hit rate and the byte hit rate [3]. This algorithm is used with the resource-based policy to make the most

use of the bandwidth and the storage space. The key idea of resource-based policy is to calculate the utilization of bandwidth and storage, and keep their utilization to be mostly the same. However, there should be enough spare storage space and bandwidth reserved for the new requests. In other words, the replacement should be started to remove the object with the least LUV before the resource runs out. A priority queue is maintained to save LUV values and hash table keys in order to remove the objects from cache proxies with the least LUV value.

Modified Least Unified Value (MLUV):
a: Constant to adjust the weight to the recent referred objects
Cavrg : Average cost during dt
Ci: Total cost during dt
dt: Time period from the last reference to the current time
n: Reference times during time period dt
Navrg: Average reference times during dt
Pn: Function value decreased with elapse time, it gives more weight to recent values.
Savrg: Average size during dt
Si: Size of caching objects
St: Total size during dt
ti: Time elapse from the moment when the LRV is calculated
Wc: Constant weight for the ratio of cost and size
Wc(i): Dynamic weight for the ratio of cost and size
Wp: Weight for popularity

$$LRV(k+1) = Pn(dt)*LRV(k) + \sum_{i=1}^{n} Wc(i) * (Ci / Cavrg) / (Si / Savrg) + Wp*((n/Navrg)/dt)$$

$$LRV(k+1) = Pn(dt)*LRV(k) + \sum_{i=1}^{n} Wc(i)* (Ci / Si) * (St / Ct) + Wp*((n/Navrg)/dt)$$

$$Wc(i) = Pn(i)*Wc$$

$$Pn(i) = e^{-a(ti/dt)}$$

This algorithm considers the historic records but need not save every historic record in comparison with the original LRV approaches. The old value is accumulated in the priority value and the weight for old values is decreased with the time elapse. To achieve better adjustment of the Modified Least Unified Value (MLUV), the formula has been normalized.

2.2.3 Prefetching and Data Coherency Algorithm

In this study, prefetching policy is applied to satisfy the requirements of Continuous Media (CM) and data coherency based on the available resources such as bandwidth and storage capacity. This policy can support the data coherency of soft real-time application and different data types such as text, image, audio and video. Also, this policy can provide predicable performance for soft real-time applications.

The resource such as storage place and bandwidth can be utilized to prefetch request objects to satisfy the data coherency of real-time data or deliver the remainder of large objects such as CM objects between the requests. During the idle times, the objects are prefetched to reduce the latency of data requests by using the available caching space and bandwidth. The available caching space and bandwidth are limited, so objects should be prefetched effectively according to the replacement priority. Replacement policy is applied to improve hit rate and byte hit rate of cached objects. If the prefetched objects are not used, the bandwidth and storage space are wasted. It is compliant with the goal of replacement policy if the objects with highest priority values are prefetched. On the other hand, the objects with the same popularity and higher cost efficiency should be prefetched. Also, the remainder of CM data will be prefetched with the available bandwidth and caching space.

2.2.3.1 Pull and Push Mechanism

As shown in Figure 2, S(t) denotes the data source value, C(t) and U(t) denote the values at the cache and at the user respectively. A constraint c is specified by the user. Servers need not inform the user if the changes of data magnitude are less than c in the source data, but need to inform the user if the change is greater than constraint c. The system must satisfy, $|U(t) - S(t)| < c$.

To improve the scalability of systems, a novel push-and-pull hybrid data broadcast scheme is proposed for wireless information networks in [1], the simulation results show that the hybrid scheme is very effective in reducing the data access time for each class of clients.

A proxy server pulls data from web servers and pushes it to users immediately. This way can reduce the data delay between the users and the cache. If communication delay can be ignored, the user data and data of the cache can theoretically be synchronized. The user constraint can be known by the cache proxy

before the stock price actually changes though user do not know how the stock price will change. Once the change in stock price is greater than the user constraint, the cache just push the updated stock price to the user.

2.2.3.2 Modified Adaptive TTL

Adaptive TTL (Time To Live) is used to keep the temporal coherency of real-time data or documents. The proxy server has a challenging issue to maintain temporal coherency with reasonable networks traffic. Take the stock trading as an example: a broker only needs the desired information from mobile device. The broker must get the stock price on time to make the decision to sell or buy. However, a broker just wants to know the change beyond some price limit like more than $0.50 or within a certain delay like 5 minutes. In this study, the coherency of real-time data will be discussed based on the previous work in [4].

To evaluate the performance of the system, the following two metrics are used:

#polling: The number of times the source is polled; an indication of the networking overhead.

VProb: Probability of a user's temporal coherency violation.

The lower the Vprob, the better performance with respect to the temporal consistency but the higher possible cost in terms of #polling.

The adaptive algorithm uses following factors to determine the new TTL.

Dynamic TTL_{dr} within Static Bounds:

$TTL_{ds} = Max(TTL_{min}, Min(TTL_{dr}, TTL_{max}))$

Adaptive TTL:

$TTL_{adaptive} = Max(TTL_{min}, Min(TTL_{max}, TTL_{dr}, TTL_{mr'}))$

$TTL_{mr'} = (f \times TTL_{mr}) + (1 - f) \times TTL_{estl}$

Fudge factor f: $0 <= f <= 1$

In this adaptive approach, dynamic TTL_{dr} is a candidate for the new TTL to reflect the changes in the new future. TTL_{mr} corresponds to the fastest source change so far and TTL_{estl} corresponds to the change trend. $TTL_{mr'}$ accommodates both of them, giving different weight to each of them by fudge factor f. If f is close to 0, it entirely relies on recent trend; this will result in a loose upper bound if the recent source changes are slow. See more details about the formula in [4].

In this study, adaptive TTL algorithm has been improved to reduce VProb and maintain acceptable #polling by adding the dynamic TTL boundary and using dynamic $VProb_{dr}$. Proxy servers poll real-time data from web server and push the data immediately to clients according to the adaptive TTL. The adaptive TTL is determined in consideration of the fastest source change rate and historic data change rate. Generally, the rapid changes of data need smaller TTL. The adaptive TTL is also bound by dynamic limits and static critical limits to avoid unreasonable network traffic or stale data due to a too small TLL or a too large TLL.

TTL_{min} and TTL_{max} are static lower bound and upper bound. They avoid unreasonable TTL in adaptive approach. They are estimated bounds in advance since the changes of stock price are unpredictable. If TTL_{min} or TTL_{max} is too low, it will cause high overheads; if TTL_{min} or TTL_{max} is too high, the high incoherency will be expected. The TTL easily becomes saturated when the data change rapidly. In this study, a modified adaptive TTL with dynamic bounds is suggested to improve the system performance.

In the modified adaptive TTL, TTL_{min} and TTL_{max} are dynamic bounds that are determined by dynamic $VProb_{dr}$ and the constraint $Vprob_{constr}$. $VProb_{dr}$ is more accurate to calculate the dynamic coherency violation and make the algorithm more adaptive. The $Vprob_{constr}$ is specified by the user in advance. Dynamic $VProb_{dr}$ is

where t1, t2 , t3,…, tn: the durations when $|U(t) - S(t)|$

$$Vprob_{dr} = \sum_{i=1}^{n} t_i \Big/ TLL_{adaptive}$$

$>= c$ during $TTL_{adaptive}$.

Rate_Polling = #Polling / $TTL_{adaptive}$

Two critical static bounds TTL_{cmin} and TTL_{cmax} are added, and dynamic TTL_{min} and TTL_{max} must change within the critical static bounds. The reduction factor of bounds is introduced to reduce the dynamic TTL_{min} and TTL_{max} according to dynamic $VProb_{dr}$, $Vprob_{constr}$ and Rate_Polling. The algorithm is as follows:

if($Vprob_{dr}$ > $Vprob_{constr}$ && Rate_Polling < Rate_Polling_factor)

{ TTL_{min} = f_red* TTL_{min};

TTL_{max} = $TTL_{adaptive}$ / f_red; }

if($TTL_{min} < TTL_{cmin}$)
$TTL_{min} = TTL_{cmin}$;
if($TTL_{max} > TTL_{cmax}$)
$TTL_{max} = TTL_{cmax}$;
where 0.0 <f_red < 1, Rate_Polling_factor > 4.

When $Vprob_{dr} > Vprob_{constr}$, the possible reason is that the $TTL_{adaptive}$ reaches the static lower bound or the rapid change of stock price needs smaller $TTL_{adaptive}$. To control the number of polling, the constraint Rate_Polling < Rate_Polling_factor is introduced, which means the #polling increasing is under control. At the same time, the TTL_{max} is determined by current $TTL_{adaptive}$ to avoid too large upper bound. Finally, the dynamic TTL_{min} and TTL_{max} are checked by the critical static bounds TTL_{cmin} and TTL_{cmax}.

3 Analysis and Discussion

The proposed proxy caching system is simulated to investigate the replacement algorithm and the adaptive TTL to improve the performance of client proxies and proxy servers. Due to the resource limitations and needs to simplify the simulation, the assumptions and simplification are made as follows:
The network bandwidth is high enough for proxies to prefetch the remainder of large objects and maintain the data coherency during idle times. This can be satisfied by increasing bandwidth, controlling Quality of Service (QOS), rejecting new requests, etc.
Some components in the simulation system are simplified as parameters. For example, the bandwidth of every request is calculated based on the assumed total bandwidth and its usage.
No error happens during the data delivery through the network. The cost of data delivery is estimated in the average value with some fluctuations.
Hash table collision does not happen when adding cache objects into the hash table or retrieving the location of cached objects in cache proxies.
Messages among the client proxies, proxy servers and Web servers can be handled by each other without errors and delays.

3.1 Evaluation of Replacement Algorithm

The simulation of proxy servers to cache requested objects was conducted using the proposed replacement and prefetching algorithm to investigate the performance of proxy servers. The web trace is used as the client request input. This trace contains a day's worth of all HTTP requests to the EPA (Environmental Protection Agency) WWW server. The request distribution is skewed and the requests from the clients reach the peak at afternoon. Client requests fluctuate greatly from 100 to 1000 times every 10 minutes. The request size ranges from hundred bytes to over 10MB.

Every record in the Web Trace includes information such as host, time stamp, request, HTTP version, response ID, object size, etc. The ULR is included in the request item, which is used as a key in the hash table maintained by cache proxies. The hash table stores the location of the cache objects containing the IP address and document path. Hashing is an implementation technique that guarantees $O(1)$ efficiency for Add, Remove, and Retrieve operations regardless of the number of objects in the collection. Because the objects are saved in a hash table without order, a ordered list is maintained to decide which object will be remove from hash table according to the priority value such as LUV for LUV replacement algorithm and time stamp for LRU replacement policy. The hash key is included in the ordered list and used to remove an element from the hash table when replacement policy is executed. The complexity of time can be $O(\log_2 n)$ which is the same as those of LUV in [3], LFU and LUV. Also, space complexity has the same order as those of LUV, LFU and LUV. This algorithm considers the historic records but need not save every historic record in comparison with the original LUV approaches. The old value is accumulated in the priority value and the weight for old values is decreased with the time elapse.

The objective of the replacement algorithm is to increase the hit rate, byte hit rate, and delay saving rate of cache objects at proxy servers. They are defined as follows:
Hit rate = Total reference times / Total request times
Byte hit rate = Total reference size / Total request size
Delay saving = Total response time of referenced objects / Total response time of requested objects

The hit rate, byte hit rate and delay saving are shown in Figures 3.3, 3.4, 3.5, respectively. The simulation results show that the modified LUV algorithm has better performance in terms of hit rate, byte hit rate, and delay saving rate. The caching space has an effect on the performance of proxy caching. The overall hit rate of MLUV changes from 25% up to 82 %, the byte hit rate increases from about 12% to 72%, and the delay saving increases from around 12% to 73 % when the caching size ranges from 50MB to 1GB.

The replacement algorithm of the proposed caching proxies should handle text, images and continuous Media (CM). If the entire contents of several long streams of video are stored in cache proxy, the cache space can be exhausted. A parameter called Maximum Object Ratio (MOR) is utilized to control the allowed space to store large objects in proxies, which is the ratio of the allowed storage space for large objects to the total cache size of proxies. In other words, a fixed storage space is allocated to save a part of large objects according to MOR. If MOR is fixed and the total cache size of proxies is larger, a bigger part of the larger object is allowed to be stored in cache proxies. The remainder of the large objects will be fetched from the data source when it is requested by the clients.

In Figures 3.6, the hit rate of the caching proxies change with the Maximum Object Ratio (MOR) when different replacement policies such as MLUV, LUV, LRU and LFU are applied. This simulation is conducted when the total cache capacity is 1 GB. The four replacement policies have the same behavior. The hit rate, the byte hit rate, and the delay saving rate decrease as the MOR increases. When the Maximum Object Ratio increases, a large cache object is allowed to be stored in the proxy server and other small cache objects are dropped out of the proxy server. The small number of cache objects may reduce the hit probability. For the MLUV algorithm, the hit rate decreases from 82% to 30%. The similar behaviors are observed with the byte hit rate, and the delay saving rate of proxy servers. It is shown that the MOR must be set within a certain range to maintain the hit rate, the byte hit rate, and the delay saving rate of proxy servers when the size of the cache objects is too large.

It is also found that the hit rate, the byte hit rate, and the delay saving rate can still be kept at a high level when the number of group proxy servers increases, even if the MOR of every server is high. Therefore, the overall performance can still be predicted in a large proxy group even when the large objects are allowed to be cached in cache proxies. This issue will be discussed in more details for client proxies.

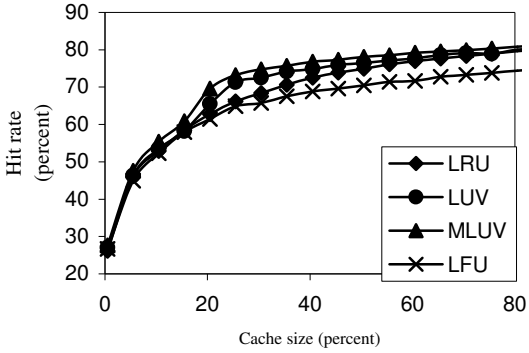

Figure 3.3: Hit rate comparison of Modified LUV(MLUV) with other cache replacement algorithms

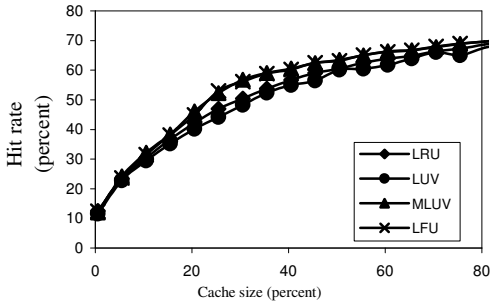

Figure 3.4: Byte hit rate comparison of Modified LRV(MLUV) with other cache replacement algorithms

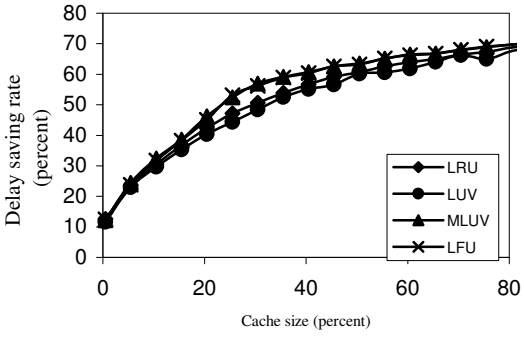

Figure 3.5: Delay saving rate comparison of modified LUV(MLUV) with other cache replacement algorithms

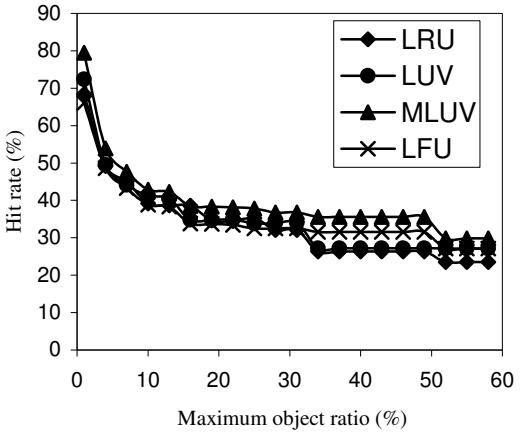

Figure 3.6: Hit rate comparison of Modified LUV(MLUV) with other cache replacement algorithms versus maximum object ratio for proxy servers

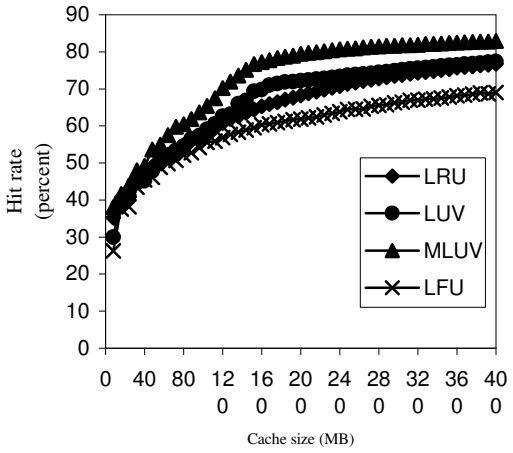

Figure 3.10: Hit rate comparison of Modified LUV(MLUV) with other cache replacement algorithms for client proxies

3.2 Evaluation of Replacement Algorithm for Client Proxies

Client proxies usually have small caching capacities but have a large amount. Simulation results show that the cooperation of client proxies can substantially improve the overall performance of the caching system in terms of the hit rate, the byte hit rate, the delay saving rate, and the computing cost. In the simulation, the client is a mobile device like a handheld with the cache capacity of 8 MB and the number of the clients can be large. The clients were grouped to share the cache objects. The weight for the ratio of cost to size is set to be 0.8 for MLUV.

In Figures 3.10, 3.11, 3.12, the hit rate, the byte hit rate and the delay saving of grouped client proxies are plotted with cache sizes respectively. The Modified Least Unified Value (MLUV) was simulated for client proxies in comparison with other replacement policies such as the Least Unified Value (LUV), the Least Recently Used (LRU) and the Least Frequently Used (LFU). The hit rate of MLUV is 5% higher than that of the LUV, 6% higher than that of the LRU, and over 13% higher than that of the LFU; the byte hit rate of the MLUV is about 3% higher than that of the LUV, 5% higher than that of the LRU, and over 11% higher than that of the LFU; the delay saving rate of the MLUV is about 3% higher than that of the LUV, 5% higher than that of the LRU, and over 12% higher than that of the LFU.

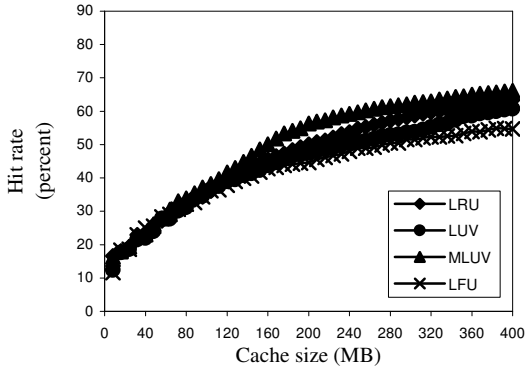

Figure 3.11: Byte hit rate comparison of Modified LRV(MLUV) with other cache replacement algorithms for client proxies

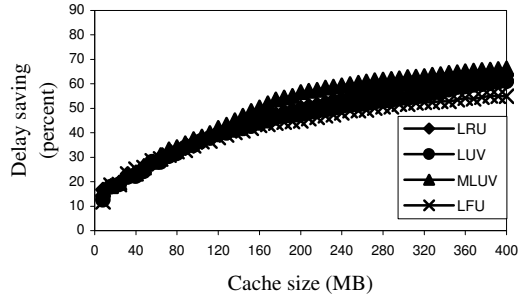

Figure 3.12: Delay saving rate comparison of Modified LUV(MLUV) with other cache replacement algorithms for client proxies

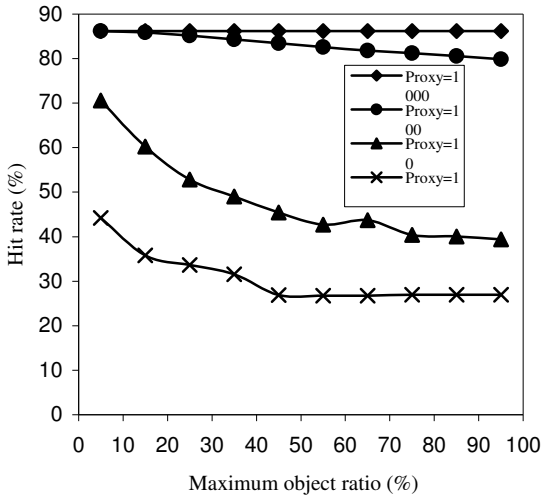

Figure 3.14: Hit rate of modified LUV versus maximum object ratio for client proxies

3.3 Data Coherency

The proxies have to handle various data types with the limited proxy space. Normally, the size of the MD ranges from several mega bytes to hundreds of mega bytes. The whole cache space can be exhausted by one large cache object like a stream of video. The caching size will be controlled for one cache object by the Maximum Object Ratio (MOR). The effect of the large objects on the performance of client proxies is simulated when the sizes of the objects range from several bytes to 50MB and the size of one single proxy is 8MB. The hit rate, the byte hit rate, and the delay saving rate of client proxies are plotted with the MOR in Figures 3.14. When the replacement algorithm is applied to a single client proxy, the hit rate decreases from 44% to 27%. Similarly, the byte hit rate drops from 23% to 12%, and the delay saving rate decreases from over 24% to 12% as the MOR changes from 5% to 95%. In other words, the allowed cache space of one cache object increases from 400KB to 7.6MB. Similarly, the hit rate, the byte hit rate, and the delay saving rate of client proxies decrease with the increment of the MOR, but the decrement of performance tends to be less obvious when the proxy number increases in the client proxy group. When the proxy number increases up to 1000, the large cache objects have less effect on the hit rate, the byte hit rate, and the delay saving of client proxies. The reason is that the small objects can be found from the other group proxies even if the large objects force a number of small objects out of the cache proxy on one host. Therefore, the performance of proxy caching is still predictable in terms of the hit rate, the byte hit rate, and the delay saving rate even if the proxies need to deal with a variety of data.

Take stock price change as an example: the simulation is implemented to investigate data coherency among the clients, proxy servers, and source servers with the adaptive TTL. As proposed in Chapter 2, the stock prices are pulled from in the Web servers and pushed to clients/users by the default proxy servers. The data delay between the cache proxy servers and the users may introduce the further delay between the stock prices presented to user and the source data. If this delay is much smaller than the TTL, the delay can be ignored.

Figure 6 shows the modified adaptive TTL algorithm has better performance in term of VPro. Figure 7 shows the #pooling of both cases, the #polling of modified adaptive TTL is from 2 to 3 times of that of original one. It is reasonable higher and has much better performance in terms of data consistency, especially when user constraint is greater than $0.3. Both Vprob value and #polling value increase since the critical minimum bound is 1min which is not small enough to meet the low user constraint. In other words, the smaller critical bound is necessary but it will cause more #polling or overheads. There is a tradeoff between the temporal coherency and system overheads.

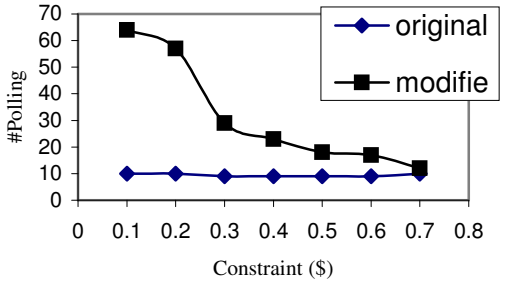

Figure 3.25: #Polling Comparison of Original and Modified Adaptive TTL
(TTL_{min} =16min, TTL_{max} = 30min, f=0.7)

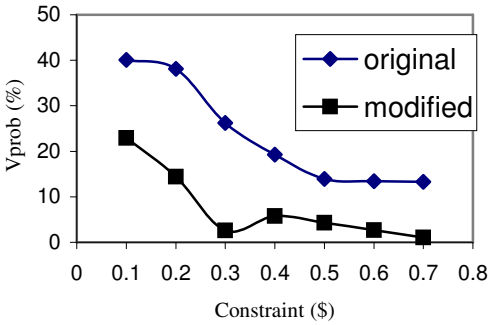

Figure 3.24: VProb Comparison of Original and Modified Adaptive TTL
(TTLmin =16min, TTLmax = 30min, f=0.7)

4 Conclusions and Future Work

The simulation of both client proxies and proxy servers was performed with different replacement algorithms such as the Least Unified Value (LUV), the Least Recently Used (LRU) and the Least Frequently Used (LFU). The simulation results show that the modified LUV algorithm has better performance in terms of the hit rate, the byte hit rate, and the delay saving rate.

Proposed cache proxies can deal with a variety of data utilizing the Maximum Object Ratio (MOR) to control the cache size for the requested large objects in a cooperative cache proxy group. The remainder of the large objects will be prefetched once the large object is requested. It is shown that the performance of proxy caching scheme is still predictable in terms of the hit rate, the byte hit rate, and the delay saving rate even if the proxies need to deal with a variety of data. However, this will put more pressure on bandwidth utilization.

The cooperative architecture allows the proxy group members to share resources with one another. This improves the reliability of the system and the predictability of the performance even when the size variance of cache objects is great. The number of the cache proxies should be controlled to avoid the need to maintain a large hash table and overwhelming network traffic among group proxies. The group proxies can communicate with the other proxies at the same level or the higher level to achieve better scalability.

A push and pull hybrid scheme for the cache proxies was applied in this study. This can solve the scalability problem and avoid making modification to the source severs and the HTTP protocols. Users can specify the constraints of data objects for a cache proxy. The users only get the information that is interesting and necessary. It will reduce unnecessary network traffic.

The modified adaptive approach with the dynamic TTL bounds is shown to solve the problem caused by the static TTL bounds through the simulation. When the domain is not well known, the static bounds may not be specified correctly. It may cause high temporal incoherency and increase overheads. The dynamic bounds are determined by the rate of polling, the dynamic Vpro, and the constraint factors. In this way, the more reasonable TTL bounds can be specified even if the initial TTL bounds are not specified appropriately. In comparison with the original adaptive TTL algorithm, the modified adaptive TTL has better adaptive performance when the data change rapidly and the user constraint is more critical. With better TTL adaptation, the modified adaptive TTL algorithm has better performance in terms of the combination of temporal coherency and system overheads.

In the future, we will attempt to fine-tune the algorithms to achieve a more predictable performance necessary for soft real-time applications.

References

[1] J. Hu et al., “A Novel Push-and-Pull Hybrid Data Broadcast Scheme for Wireless Information Networks,” *Proceedings of IEEE 2000*, 2000.

[2] T. M. Kroeger et al., “Exploring the Bounds of Web Latency Reduction from Caching and Prefetching”, *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems*, Monterey, CA, December 1997.

[3] H. Bahn et al., “Efficient Replacement of Nonuniform Objects in Web Caches,” *Computer*, June 2002.

[4] Raghav Srinivasam, Chao Liang, and Krithi Ramamritham. Maintaining Temporal Coherency of Virtual Data Warehouses, in proceedings of IEEE 1998, 1998.

[5] P. Rodriguez, C. Spanner, and E. W. Biersack, “Web Caching Architectures: Hierarchical and Distributed Caching, ” *Proceedings of WCW’99*, 1999

[6] E. Bommaiah et al., “Design and Implementation of a Caching System for Streaming Media over the Internet,” *IEEE Real Time Technology and Applications Symposium*, June 2000.

[7] C.M. Bowman et al., “The Harvest Information Discovery and Access System,” *In Second International World Wide Web Conference*, October 1994.

[8] R. Caceres et al., "Web Proxy Caching: The Devil is in the Details," *ACM, Performance Evaluation Review*, 26(3): pp. 11-15, December 1998.
[9] R. P. Wooster and M. Abrams, "Proxy Caching that Estimates Page Load Delays," *Proceedings of the 6th International WWW Conference*, April 1997.
[10] M. Rabinovich et al., "Not All Hits are Created Equal: Cooperative Proxy Caching Over a Wide Area Network," *In 3rd International Web Caching Workshop*, June, 1998.
[11] J. Yang et al., "Dynamic Web Caching," *In Tech Report*, University of California, Los Angeles, November 1998.
[12] R. Rajaie, J. Kangasharju, "Mocha: A Quality Adaptive Multimedia Proxy Cache for Internet Streaming," *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Port Jefferson, New York, June 2001.