Performance Improvement of Graceful Image Caching by Using Request Frequency Based Prefetching Algorithms

Zhou Su, *Nonmember,* Teruyoshi Washizawa, Jiro Katto and Yasuhiko Yasuda, *Members, IEEE*

***Abstract*--This paper presents a cache scheme for image databases, web browsers, proxies and other similar applications in Internet. The current cache system employs a hard strategy: either the image is stored in a cache or not even its data is quite big. We have proposed a caching scheme featuring human memory mechanism (image database system featuring graceful oblivion), which is called Graceful Caching. The previous work shows that Graceful Caching reduces user response time by assigning a variable amount of memory to each image. Several recent studies suggest that prefetching techniques could be employed to further improve the cache performance, by anticipating and prefetching future client requests. Some approaches based on request frequency have been proposed to the conventional caching (hard system) and got better results. In this paper, we try to apply a request frequency-based approach to Graceful Caching system, in which a progressive image format is used. Based on the analysis of web request probability in Graceful Caching, we go on to propose a new prefetching algorithm that drastically improves cache performance of system. We verify the performance of this algorithm by simulations.**

***Index Terms*-- Graceful Caching, Hierarchical Image Coding, Prefetching, Web Caching**

I. INTRODUCTION

Proxy server systems are commonly introduced to reduce data traffic and improve response time in Internet, because data is cached temporarily in the intermediate server and utilized many times. However, current system employs a "hard" caching strategy: an image is stored in a cache or not even if its data is quite big.

Hierarchical coding is playing an important role to overcome this problem. Hierarchical image format has been widely used since the first hierarchical coding method was proposed[1]. Many other kinds of hierarchical formats were also designed such as pyramids [2], wavelets [3] and subband coding [4].

Based on the above background, we have proposed a caching scheme called Graceful Caching for image database system [5], in which a variable amount of memory is assigned to each image by using hierarchical coding format. Moreover, we also proposed a proxy server system with hierarchical image caching in which Graceful Caching scheme was applied to the Web [6]. Many researches paid attention to caching algorithms for this Graceful Caching [7] [8] [13]. However, most algorithms simply concern themselves with just how to remove the object from the cache. They show little consideration for how to fetch the object from the server to the cache. Recent studies suggest that prefetching technique further increases the cache hit ratio as long as future client requests are tactfully anticipated. [10][16] [17].

Top-10 approach is one of prefecthing schemes based on web request frequency. It has been proved that most of web requests to a server are for a very small set of objects [12][15], for example top 10 %. Top-10 approach defined this set of objects as 'Top-10' and only considered prefetching them [10]. This approach has been used in the conventional (hard) caching system and has better results.

In this paper, we first apply this Top-10 approach to Graceful Caching system, in which a progressive JPEG format is used. In our preliminary experiments, when image *i* is one of Top_10 popular images, one more layer of this image is prefetched. However, it achieved only slight improvement on the cache hit ratio. More so, the simulation result is not always good when the related parameters are varied.

Therefore, based on the probability analysis of web request, we then quantitatively clarifies that the more layers of the hierarchically coded image that the user has already got, the lower the probability that this user still wants to get the next layer will be. Consequently, we try to assign different lowest ranking of images to be prefetched (T_k) adaptively when the user is asking for a different layer (k th) of the image. We derive a relationship, $T_{K+1} = T_K \cdot \sqrt[\alpha]{P}$, where α is a parameter of Zipf_like distribution and p, called Greedy Degree first introduced by one of the authors [13], is a probability that the user still wants to get the $(j+1)$ th layer after having got the j th layer of this image. This prefetching policy chooses the ranking threshold adaptively according to different situations. The simulation results show the proposed one is a more practical and robust method than Top-10 prefetching no matter how the cache size and Greedy Degree are changed.

II. OVERVIEW OF GRACEFUL CACHING

A. Graceful Caching: Image Database System Featuring Graceful Oblivion

The proposed database system is based on Graceful Caching we have studied. Fig.1 shows a simple structure of distributed databases. Terminals get image data from one of center

The Authors are with the Graduate School of Science and Engineering, Waseda University, 3-4-1, Ohkubo, Shinjuku-ku, Tokyo 169-0072, Japan (E-mail:{suzhou,washi,yasuda}@yasuda.comm.waseda.ac.jp,katto@katto.comm.waseda.ac.jp).

databases via a sub-database (sub-node). If the sub-database cached the required data, terminals don't need to access the center server directly. After all, caching popular data in the sub-database leads to reducing the load of center servers, the network traffic and the response time.

The Graceful Caching system utilizes hierarchical property of images for using efficiently the memory in the sub-database. The size of image cached in the sub-database is reduced each time when a given period of time has elapsed from last access to the data (graceful oblivion). If the user is not satisfied with the resolution of the image in the sub-database, the lacking data are replenished from the center database (recollection).

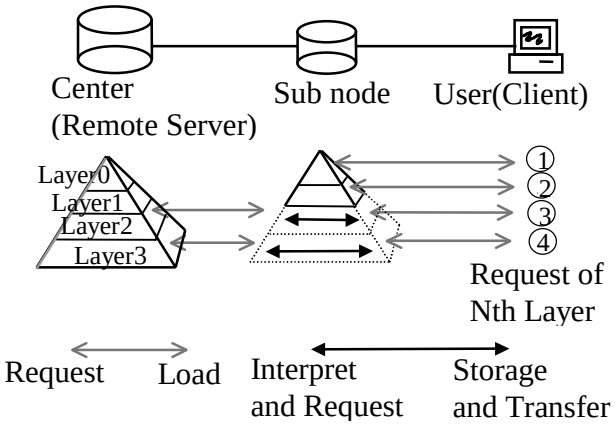

Fig.1: Image Database System with Graceful Caching

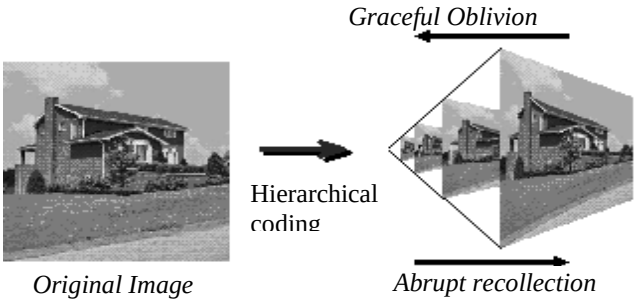

Fig.2: Image of Pyramidal Structure, Oblivion, Recollection

Fig.2 shows an example of a graceful oblivion and recollection process in the image of pyramidal structure. By introducing such an image storage system, the number of images cached in a sub-database is much increased and the cache hit ratio is greatly improved as compared with conventional storage systems.

B. Proxy Server System with Hierarchical Image Caching

Conventional cache management is ``hard" because objects are either present in the cache or not. Graceful Caching, however, is more soft and flexible, in which variable amounts of cache memory can be assigned to each image. Therefore assuming that a progressive image format is used, a low resolution version of the original image can be kept in the cache. The main advantage of this approach is that, in many cases, users will not need to download the full resolution image from the server.

Based on this background, we applied the theory of graceful image database system to proxy server and proposed a new caching scheme called Graceful Caching system [6][13], which allows some of the most popular web objects to be cached at intermediate nodes. And network has been shown to provide substantial performance improvement.

III. Replacing Policies for Graceful Caching System

A. Replacing policy

Many algorithms address the issue about replacing policy in Graceful Caching system. When a new object comes in and the cache exceeds its limit, one or more object in the cache must be removed in order to make room for the newly accessed object. Replacing algorithm is to decide which object should be removed.

Although the purpose of this paper is not to discuss the replacing policy, we briefly introduce one called TTLLRU [13]. We know LRU, abbreviation of the phrase “Least Recently Used”, is a very popular replacing algorithm. The least recently requested object is first removed from the cache according to LRU. TTL means ”Time to Live’’. TTLLRU replacing policy is specially proposed to Graceful Caching system after absorbing the idea of LRU.

We assign a variable TTL to each layer of the images as the following formula.

$$TTL = (n \times a) - (t_{now} - t_0) \quad (1)$$

Here, n denotes the request times. a is the parameter of TTL. t_{now} denotes the current time. t_0 is the time when the object was taken from the server into the cache. If the request frequency of a certain layer gets up or it is requested more recently, the variable TTL of that layer will increase.

In TTLLRU algorithm, we will firstly compare the TTL value of the lowest layer (biggest size layer) of every object in the cache. Then remove the layer that has the smallest TTL value.

B. Drawbacks of Simple Replacing Policy

Replacing policy has been approved to be useful by many papers. But most algorithms simply concern about just how to remove object from the cache [7][8] [9]. We call them simple replacing algorithms because they show little consideration for fetching object from the server into the cache.

For example, when a user requests for any layer of the image and miss hit happens at the same time, cache will just fetch that absent layer from the server. This policy is simple and useful if the user won’t continue to request for the next layer of the same image. Otherwise, the user must wait for long time because the cache need to fetch the absent layer from the server one layer by one layer after getting the request from the user. As a result, we think fetching policy should be needed when the user always wants to get the more layers of the same image. If we can predict user’s future requests and prefetch the data needed, the cache performance must be improved.

IV. Prefetching Policies for Graceful Caching System

A. Prefetching Policy Based on Top-10 Approach

Top-10 approach is one of the newest policies and it has showed better performance in hard caching system. Studies proved that large percentage of web requests are for a small set of objects [12][16]. Top-10 approach prefetches only the most popular objects (This set of objects is called Top-10 where the name comes from) and only to clients that will be able to use them. The actual number of objects in the Top-10 is fine-tuned based on user profiles and the amount of cache space. Now we try to apply Top-10 approach to Graceful Caching. Here, both replacing policy and prefetching policy will be used. The replacing policy is the same as TTLLRU that we have talked about.

When miss hit happens and the user is asking for the k th layer of image i which ranking is r_i according to its request times, if the following conditions are satisfied:

$$u_i < k < 4 \qquad (2)$$
$$r_i < T$$

The cache will fetch k th layer and prefetch $(k+1)$ th layer from the server.

Here, we will explain Equation (2). We make an assumption that the hierarchical image has 4 layers in total according to Progressive JPEG format. As for image i, only the top u_i layers of this image are stored in the cache. T is a threshold of ranking based on Top_10 approach. Image i is ranked as the r_i th most popular image according to its access frequency. If image i is one of Top_ T popular images, one more layer of this image will be prefetched.

B. Proposed Prefetching Policy

Top-10 approach can get better result for cache performance than the policy without prefetching. However, it achieved only slight improvement on cache hit ratio. Moreover, the simulation result seems not to be always good when the cache size or Greedy Degree is variable

Based on the analysis of web request probability in Graceful Caching, we go on to propose a new prefetching algorithm that is more practical for this system.

This section is organized as follows. First, we analyze web request distribution in Graceful Caching and get a conclusion of request probability. Secondly, after theoretical analysis we find the relations among ranking thresholds for different layers. Lastly, we will introduce the proposed algorithm.

1. Web request distribution in Graceful Caching

In hard caching, if the ranking of image i is r_i according to its request times, the probability of a request for this object is $P(i) = \frac{\Omega}{r_i^{\alpha}}$ (3). α is a parameter of Zipf_like distribution and it varies from trace to trace, ranging from 0.64 to 0.83.[11][12].

In Graceful Caching, the probability of the request for image i is the same. However, this probability includes the probability of the request for every layer. Moreover, the probability of the request for a certain layer of one image is still unknown. Here, This paper tries to find the probability that the user requests for the j th layer of image i in Graceful Caching.

We define the probability space on a set of N images with l layers, $Q=\{A, B, P\}$ as follows:

$A=\{ a_{i,j}$; the j th layer of image $i\}$,

$i=\{1,\ldots,N\}$, $j=\{1,\ldots, l\}$,

$B=\{ b_{i,j} = \{ a_{i,1}, a_{i,2}, \ldots., a_{i,j} \}\}$,

$P: B \rightarrow [0,1]$

Here, B is Borel field, the family of partial subsets of A which is strictly regulated by Progressive JPEG encoding format. For example, the user can't request for the $(j+1)$ th layer of one image unless he has got the j th layer before [14]. P is the probability measure on B, $P(b= b_{i,j}) = P(i, j)$.

We can obtain the probability of the request for the j th layer of image i as the following equation:

$$P(b_{i,j}) = P(i, j) = \frac{p^{(j-1)} \cdot \Omega}{(\sum_{m=0}^{l-1} p^m) \cdot r_i^{\alpha}} \qquad (4)$$

Please see Appendix for Proof.

2. Theoretical Analysis

Let us talk about the probability of the request for the j th layer of image i in Equation (4).

Here, we will explain more about the above equation.

The value of $P(i, j)$ will decrease when j increases. It can be easily proved as follows.

Let $P(i, k)$ be the probability of the request for the k th layer of image i and $P(i, q)$ be the probability of the request for the q th layer of image i. Here $k > q$ and $k - q \geq 1$.

From Equation (4), we can get

$$\frac{P(i,k)}{P(i,q)} = \frac{p^{(k-1)} \cdot \Omega / (\sum_{m=0}^{l-1} p^m) \cdot r_i^{\alpha}}{p^{(q-1)} \cdot \Omega / (\sum_{m=0}^{l-1} p^m) \cdot r_i^{\alpha}} = p^{k-q}$$

Since $k - q \geq 1$ and $p \leq 1$, then $P(i,k) < P(i,q)$. We can predict such phenomenon. The more layers that the user has already got, the lower the probability that this user still wants to get the next layer will be. So we should assign different ranking threshold to different layer.

For example, when a user is asking for the first layer of image i which ranking is r_i, we set up the threshold as T_1. If $r_i < T_1$, we will prefetch both the first layer and second layer. When a user is asking for the second layer of image i, we set up the threshold as T_2. If $r_i < T_2$, we will prefetch both the second layer and third layer. Since $P(i, 2)$ must be smaller than $P(i, 1)$, it can be expected that T_2 should be smaller than T_1. The reason is that we only want to prefetch the object that has high probability to be requested by the user later. As a result, we try to assign different ranking threshold (T_K) when user is asking for different layer (k th) of image i.

However, how do we find the relations among different threshold T_K?

If we assume $P(i(T_K), k)$ is constant to any k, we need to find $P(i(T_{K+1}), k+1)$ has the same probability as $P(i(T_K), k)$. Here,

i(T_i) is the index of the image of which ranking is T_i. We have showed that

$$P\,(i(T_K),\ k) = \frac{p^{(k-1)} \cdot \Omega}{(\sum_{m=0}^{l-1} p^m) \cdot (T_K)^{\alpha}}$$

$$P\,(i(T_{k+1}),\ k+1) = \frac{p^{k} \cdot \Omega}{(\sum_{m=0}^{l-1} p^m) \cdot (T_{K+1})^{\alpha}}$$

When P(i(T_K), k) is equal to P(i(T_{K+1}), k+1) ,we can get

$$T_{k+1} = \sqrt[\alpha]{p} \cdot T_k \qquad (5)$$

The probability that the user asks for the *(k*+1*)* th layer of the T_{K+1}'th most popular image is equal to the probability that the user asks for the k th layer of the T_K th most popular image.

3. The Proposed Policy

Based on the theoretical analysis in the previous section, we will use different ranking threshold (T_K) when the user asks for the different (k th) layer.

For example, When the user is asking for the image i from the first layer, cache will prefetch both the first layer and the second layer of image i from the server if $r_i < T_1$. Otherwise ($r_i >= T_1$), cache will only fetch the first layer.

After using the probability analysis, we can get the following fetching policy.

When miss hit happens after the user has asked for the k th layer of image i at that time, if the following conditions are satisfied:

$$\begin{aligned} & u_i < k < 4 \\ & r_i < T_K \\ & T_K = T_{K-1} \cdot \sqrt[\alpha]{p} \end{aligned} \qquad (6)$$

the cache will not only fetch the k th layer but also prefetch the *(k+1)* th layer from the server.

Here, we will explain the parameters in Equation (6).

1) Each image has 4 layers in total according to Progressive JPEG format. As for image i, only the top u_i layers of the image are stored in the cache.

2) α is the parameter of Zipf_like distribution.

3) p is Greedy Degree, which means the probability that the user still wants to get the (k+1)th layer after having got the k th layer of the image.

V. Evaluation of Algorithms

A. Simulation Conditions

We assume that there are 10000 different images in the server, with their own ranking r_i . The images are supposed to be accessed according to Zipf-like frequency distributions [12]. In Ref. [10], the actual number of images in the group of Top-10 (which is not always 10) is fine-tuned based on the different situation. But we still set the threshold as 1000 (10% of the total images) in the simulation as a normal situation. As for the proposed prefetching policy, we set the threshold of the first layer as $T_1 = N \cdot \sqrt[\alpha]{P}$, where N is the total number of the images in the server. The original images will be encoded into a hierarchical image by four layers. The relation among the size of different layer is proportional to 5:13:22:59.The fourth layer has the largest size.

B. Evaluation Criteria

Hit-ratio is defined as the ratio of request times that the data requested by the user can be directly found in the cache to the total number of requests from the user.

Byte hit-ratio is defined as the ratio of the data size when the data requested by the user can be directly found in the cache to the total data size requested by the user.

Hit-ratio and Byte hit-ratio have been popularly used to evaluate the cache performance by most researchers [7][9] [10] [13]. When the ratio is sufficiently high, the user can get the data from the local cache directly, the response time can be remarkably reduced.

C. Simulation Results

In this section we will experimentally compare the performance of three caching algorithms: the simple TTLLRU algorithms without prefetching, Top_10 prefetching and the proposed prefetching in this paper.

1) Several researchers have observed that the distribution of web request from a fixed group of users follows a Zipf-like distribution, $\frac{\Omega}{r_i^{\alpha}}$. Besides, the value of α , a parameter of Zipf-like distribution, varies from trace to trace, ranging from 0.64 to 0.83 [12].

We firstly tested the cache performance with respect to the varied parameter, α . Here, the simulation time is 200 days. For each of these cache policies, we considered the cache size equal to 5% of the total size of all images in the server. Moreover, Greedy Degree is chosen to be 0.4.

Fig.3 shows the hit ratio of the cache when we use the different policies. Both proposed prefetching policy and Top_10 prefetching policy can get better result on the hit ratio. However, Top_10 prefetching achieves only slight improvement while the improvement of the proposed one is quite good.

Fig.4 contains byte hit ratio attained by the different policies for different simulation time. Proposed prefetching policy has the greatest improvement which is around 10%.

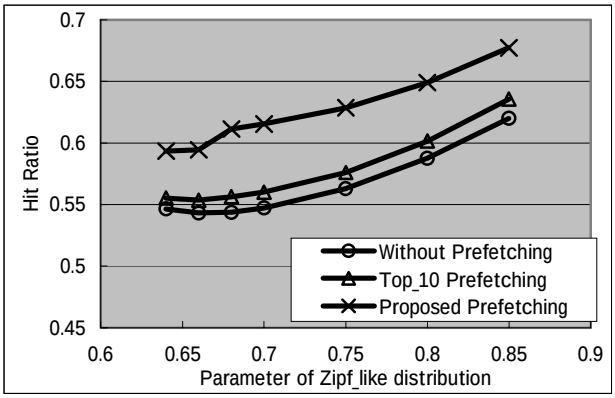

Fig.3: Hit Ratio vs. Parameter of Zipf-like Distribution

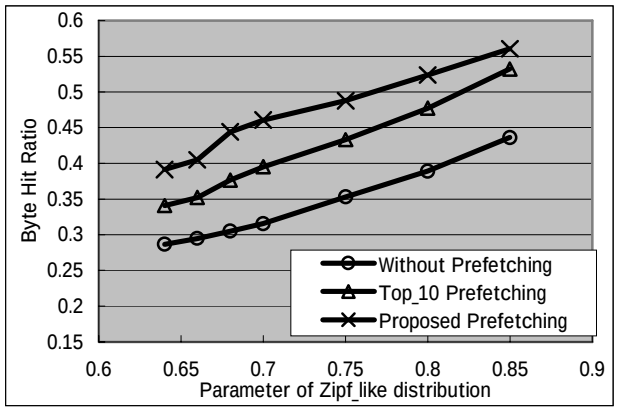

Fig.4: Byte Hit Ratio vs. Parameter of Zipf-like Distribution

2) An important measure is the robustness of the various policies. It is well known that the performance of the prefetching policy depends upon whether clients always request the next layer of the current image, which is called Greedy Degree. For example, the Top_10 prefetching must be weak if the user never requests the next layer. As a result, it is necessary to see how Greedy Degree (*p*) factors into the robustness of the different policies.

Fig.5 shows the relative hit ratio with respect to the original method without any fetching. Therefore, the relative hit ratio of the policy without prefetching is always one. For a given policy, we expect this relative ratio to be larger than 1.

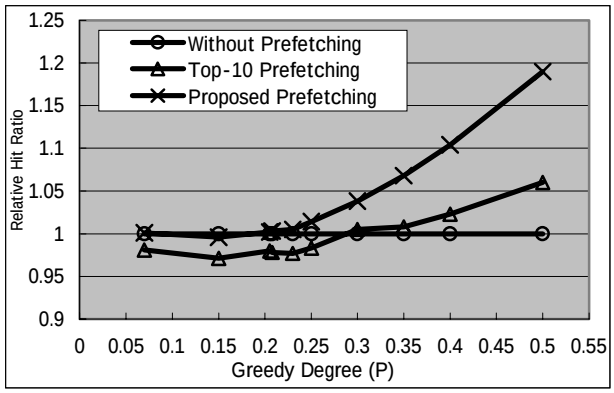

Fig.5: Relative Hit Ratio Compared with the Method Without Prefetching

As we see from Fig.5 and Fig.6, both Top-10 prefetching and proposed prefetching get better result when *p* increases. And the proposed prefetching always outperforms Top_10 prefetching at the same value of *p*. However, the relative ratio of the Top_10 prefetching is smaller than one when *p* is very small. That is, Top_10 prefetching performs very poorly when Greedy Degree (*p*) is low.

This result can be understood. When *p* is very small, the user seldom requests for the next layer of the current image. To prefetch the next layer of all Top-10 images is not worth. This policy can outperform the policy without any prefetching only in the situation that *p* is high. From Fig.5 and Fig.6, we can get the followed situation: Top-10 gets better performance on hit ratio when *p* is higher than 0.3 and better performance on byte hit ratio when *p* is higher than 0.1.

Top-10 prefetching is worse on hit ratio than on byte hit ratio when *p* is small. We know the next layer has bigger size than the current layer in hierarchical coding. When *p* is very small, the cache prefetches big size object and takes big space of its. However, these big size objects are seldom requested by the user as *p* is small. Less space is left in the cache for the newly coming object. So the hit ratio becomes worse.

We observed that Proposed prefetching works better. Let's look back to the threshold of proposed prefetching policy. As we can see from Equation (6), the threshold of this policy is determined not only by Zipf'like parameter but also by *p*. When *p* is small, the threshold is also small. For example, when *p* is zero, the threshold of the proposed policy is also zero. We will prefetch nothing at this time in case of the proposed one. But the threshold of Top-10 is still kept as the same value. On the other hand, when *p* is one, the threshold of the proposed policy is the number of total images. But the threshold of Top-10 is still not changed. We think the threshold of Proposed policy is an adaptive one. However, the threshold of Top-10 policy is a still one. So we think the former is more robust to varying situation when Greedy Degree (*p*) is changed.

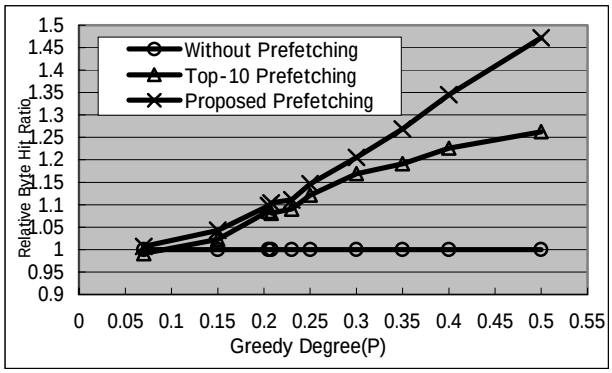

Fig.6: Relative Byte Hit Ratio Compared with the Method Without Prefetching

3) Another important parameter in prefetching algorithm is the cache size. The cache performance also depends on the cache size relative to the size of all images. Here, We compare the performance of three different caching policies. For each of these cases, Greedy Degree is chosen to be 0.4. Moreover, the simulation time is 200 days.

Fig.7 and Fig.8 show the hit ratio and byte hit ratio with respect to the cache size. Proposed prefetching policy consistently outperforms the simple replacing policy and Top_10 prefetching, but the gain drops as the relative cache size increased. The intuition behind this is that at this point the cache size is big enough to hold the set of most popular images that could satisfy almost all requests. However, this situation can be found in real system hardly since the cache size is always limited in most cases, we think prefetching policy is still a practical one

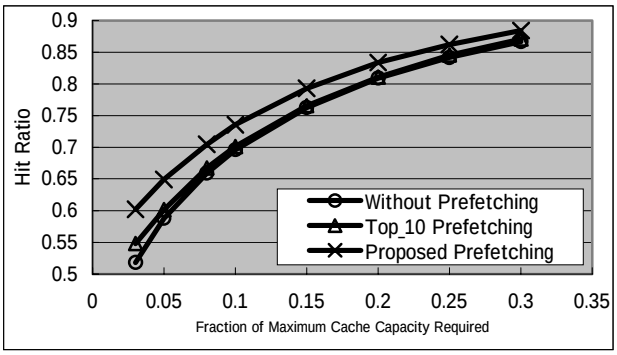

Fig.7: Hit Ratio vs. Cache Size

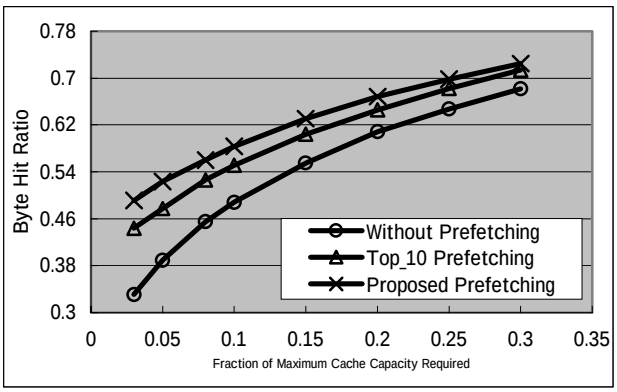

Fig.8: Byte Hit Ratio vs. Cache Size

4) Finally user response time is addressed. Simulations are carried out under the same condition stated in Section 3), where Greedy Degree is chosen to be 0.4 and simulation time is 200 days. Moreover, the cache size is equal to 5% of the total size of all images in the server. According to the investigation on the range of the users' access bandwidths shown in Ref. [7], we assume the bandwidth between clients and proxy to be 1Mbps and the bandwidth between servers and proxy to be 64Kbps.

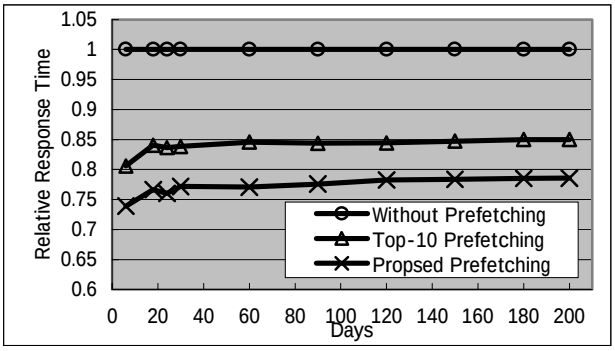

Fig.9: Response Time vs. Simulation Time

Fig.9 shows the relative response time with respect to the original method without any fetching. The results show that the proposed prefetching algorithm has the shortest relative response time. It can reduce the response time up to 23 %, which is almost 7 % more than the reduction that Top_10 prefetching has achieved. The accordance among the results in Fig.3, Fig.4 and Fig.9 also verified that the proposed algorithm obtains shorter response time as well as higher hit ratio and byte hit ratio.

VI. Conclusion and Further work

In this paper, we discussed prefetching algorithms in Graceful Caching system. We firstly applied Top_10 approach to Graceful Caching system. Based on the mathematical analysis of web request probability, we then proposed a new prefetching algorithm to overcome disadvantages of Top_10 based approach.

We compared the proposed algorithm with the previous ones from the viewpoint of the hit ratio and byte hit ratio by simulations. Based on these experimental results, we conclude that the proposed prefetching algorithm drastically outperforms previous ones. Better performances on the hit ratio and the byte hit ratio are achieved and it was also shown that the proposed algorithm was much robust to varying Greedy Degrees and cache sizes.

As for the future research, several problems still remain. These includes theoretical analysis based on a more realistic model, the design for the optimal value of Greedy Degree, network traffic estimation and the application to the moving pictures. These investigations are being performed and the results will be reported hereafter.

Appendix

Web Request Distribution in Graceful Caching

In Graceful Caching, the probability of the request for image *i* is the same as the hard caching, which can be found in Equation (3). But this probability includes the probability of the request for every layer. Here, This paper wants to get the probability that the user requests for the *j* th layer of image *i* in Graceful Caching.

Assuming that r_i is the ranking of image *i* according to its request times, we obtain the probability of the request for the *j* th layer of image *i* as the following equation:

$$P(b_{i,j}) = P(i, j) = \frac{p^{(j-1)} \cdot \Omega}{(\sum_{m=0}^{l-1} p^m) \cdot r_i^{\alpha}} \quad (4)$$

Proof:

1) As we know, the probability of the request for image *i* in Graceful Caching is the same as hard caching. But this probability includes the probability of the request for the every layer of image *i*.

$$P(b_i) = P(i) = \sum_{j=1}^{l} p(b_{i,j}) = \sum_{j=1}^{l} p(i, j)$$

Here, $P(i)$ is the total probability of the request for image *i*. $P(i, j)$ is the probability that the user requests for the *j* th layer of image *i*. That is to say: The user is not satisfied with the quality of (*j*-1) th layer, he wants to get the top *j* layers of image *i*. Here we assumed the first layer has the smallest size and the lowest

resolution.

2) We defined Greedy Degree (p) as the probability that the user requests for the (j+1) th layer after getting j th layer. [13]. Then we can get the conditional probability $b_{i,j+1}$ given $b_{i,j}$:

$$P(b_{i,j+1} \mid b_{i,j}) = p .$$

As the user always gets the image one layer by one layer gradually in Graceful Caching, the user can't get the (j+1)th layer before he has already got the j th layer [14]. That is:

$$P(b_{i,j+1} \mid B \setminus \{b_{i,j}\}) = 0 .$$

As ($B \setminus \{b_{i,j}\}$) is the complement of { $b_{i,j}$ }, we get:

$$P(b_{i,j+1}) = P(i, j+1)$$

$$= P(b_{i,j+1} \mid b_{i,j}) \cdot P(b_{i,j}) + P(b_{i,j+1} \mid (B \setminus \{b_{i,j}\})) \cdot P(B \setminus \{b_{i,j}\})$$

$$= p \cdot P(b_{i,j}) + 0 \cdot P(B \setminus \{b_{i,j}\})$$

$$= p \cdot P(b_{i,j}) = p \cdot P(i, j)$$

$$P(b_{i,j}) = p^{j-1} \cdot P(b_{i,1}) \qquad (A\text{-}1)$$

3) We can get the following function:

$$P(b_i) = p(i) = \sum_{j=1}^{l} P(b_{i,j})$$

$$= P(b_{i,1}) + p \cdot P(b_{i,1}) + p^2 \cdot P(b_{i,1}) + \ldots + \cdot p^{l-1}(b_{i,1})$$

$$= (\sum_{m=0}^{l-1} p^m) \cdot P(b_{i,1})$$

From Equation (3) we can get:

$$\frac{\Omega}{r_i^{\alpha}} = (\sum_{m=0}^{l-1} p^m) \cdot P(b_{i,1})$$

$$P(b_{i,1}) = \frac{\Omega}{(\sum_{m=0}^{l-1} p^m) \cdot r_i^{\alpha}} \qquad (A\text{-}2)$$

Eq.(4) can be obtained by substituting Eq.(*A-2*) into Eq.(*A*-1). So, we can get the probability that the user requests the *j*'th layer of image *i*.

4) Now we will prove that the sum of the total probability of the request for every layer of *N* images is one.

$$\sum_{i=1}^{N}\sum_{j=1}^{l} P(i, j) = \sum_{i=1}^{N}\sum_{j=1}^{l} \frac{p^{(j-1)} \cdot \Omega}{(\sum_{m=0}^{l-1} p^m) \cdot r_i^{\alpha}}$$

$$= \sum_{i=1}^{N} \frac{\Omega}{r_i^{\alpha}} \sum_{j=1}^{l} \frac{p^{(j-1)}}{\sum_{m=0}^{l-1} p^m} == \sum_{i=1}^{N} \frac{\Omega}{r_i^{\alpha}} (\frac{1 + p + \ldots + p^{l-1}}{1 + p + \ldots + p^{l-1}}) = 1$$

REFERENCES

[1] Y. Yasuda, M. Takagi, S. Kato, and T. Awano, "Step by step transmission and display of still pictures by a hierarchical coding method", IEICE Trans. Commun., Vol.J63-B, no.4, pp.379-386, April 1980

[2] P. J. Burt and E. H. Adelson, "The Laplacian pyramid as a compact image code", IEEE Trans. Commun., Vol.31, no.4 pp.532-540, April 1983

[3] I. Daubechies, "Orthogonal bases of compactly supported wavelets", Comm. Pure Appl. Math., pp.909-996, Nov 1988

[4] M. Vetterli, "Multi-dimensional subband coding: some theory and algorithms", Signal Processing, 6, pp.97-112, April 1984

[5] Y. Yasuda, T. Yasuno, F. Katayama, T. Toida, and H. Sakata, ''Image database system featuring graceful oblivion", IEICE Trans., Commun., Vol.E79-B, No.8, pp.1015-1021, August 1996

[6] K. Kamogawa, D. Nakajima, and Y. Yasuda, "Proxy server systems with hierarchical image caching", IIEEJ, Vol.27 No.5, pp.548-556, Oct 1998

[7] A. Ortega, F. Carignano, S. Ayer, and M. Vetterli, "Soft Caching: Web cache management techniques for images", IEEE Signal Proc. Society Workshop on Multimedia Signal Processing, (Princeton, NJ), June 1997.

[8] X. Yang and K. Ramchandran, "An optimal and efficient soft caching algorithm for network image retrieval," in Proc. of ICIP, Chicago, IL, Oct 1998

[9] M. Arlitt, R. Friedrich, and T. Jin: "Performance evaluation of web proxy cache replacement policies'', Lect. Notes Computer Sience, Vol.1469, pp.193-206 1998

[10] E. Markatos and C. E. Chronaki: "A Top-10 approach to prefetching on the Web'' In Proceedings of INET' 98, Geneva, Switzerland, July 1998.

[11] V. A. F. Almeida, M. A. G. Gesrio et al" Analyzing the behavior of a proxy server in the light of regional and cultural issues"1998

[12] L. Breslao, P. Cao, L. Fan, G. Phillips, and S. Shenker "Web caching and Zip-like distributions: Evidence and implications" Proc. IEEE INFOCOM'99

[13] T. Ishikawa, W. B. Hui, H. Ohsawa, and Y. Yasuda "A method of improving response time in still database based on CD-ROM changers by Graceful Caching'' IIEEJ, 1999,Vol 28,No 5 pp.605-611

[14] W. Pennebaker and J. Mitchell, "JPEG Still Image Data Compression Standard''.Van Nostrand Reinhold, 1994

[15] J. Gwertzman. "Autonomous Replication in Wide-Area Networks". Technical Report 17-95, Harvard University, 1995

[16] L. Fan, Q. Jacobson, P. Cao and W. Lin "Web Prefetching Between Low-Bandwidth Clients and Proxies: Potential and Performance" in 1999 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems May, 1999

[17] K. M. Koeger, D. D. E.long, and J .C .Mogul, "Exploring the bounds of web latency reduction from caching and prefetching". in Proceedings USENIX Symposium on Internet Technology and System, Dec 1997