PAPER

Integrated Pre-Fetching and Replacing Algorithm for Graceful Image Caching

Zhou SU[†a)], ***Student Member***, **Teruyoshi WASHIZAWA**[††], **Jiro KATTO**[†], ***Regular Members***, ***and*** **Yasuhiko YASUDA**[†], ***Honorary Member***

SUMMARY The efficient distribution of stored information has become a major concern in the Internet. Since the web workload characteristics show that more than 60% of network traffic is caused by image documents, how to efficiently distribute image documents from servers to end clients is an important issue. Proxy cache is an efficient solution to reduce network traffic. And it has been shown that an image caching method (Graceful Caching) based on hierarchical coding format performs better than conventional caching schemes in recent years. However, as the capacity of the cache is limited, how to efficiently allocate the cache memory to achieve a minimum expected delay time is still a problem to be resolved. This paper presents an integrated caching algorithm to deal with the above problem for image databases, web browsers, proxies and other similar applications in the Internet. By analyzing the web request distribution of the Graceful Caching, both replacing and pre-fetching algorithms are proposed. We also show that our proposal can be carried out based on information readily available in the proxy server; it flexibly adapts its parameters to the hit rates and access pattern of users' requesting documents in the Graceful Caching. Finally we verify the performance of this algorithm by simulations.

key words: *graceful caching, hierarchical image coding,content delivery network, web performance, caching algorithm*

1. Introduction

As the scale and use of the Internet increase, Web content providers find that it is difficult to serve all users with low response time, especially in the face of high loads. In recent years, how to set up contents distribution networks to efficiently distribute stored information has become a major concern in the Internet [21]–[23].

Since the web workload characteristics shows that almost 67% of network traffic is caused by image documents [7], how to efficiently distribute image documents from servers to end clients is a very critical issue. Proxy caches are commonly introduced between servers and clients to reduce data traffic and improve response time in the Internet, because data is cached temporarily in the proxy cache and utilized many times. However, the current system employs a "hard" caching strategy: an image is either stored in a cache or not.

Manuscript received April 19, 2002.

Manuscript revised November 22, 2002.

[†]The authors are with the School of Science and Engineering, Waseda University, Tokyo, 169-0072 Japan.

[††]The author is with the Canon Research Center, Canon. Inc., Atsugi-shi, 243-0193 Japan.

a) E-mail: suzhou@yasuda.commm.waseda.ac.jp

Hierarchical coding is playing an important role in overcoming 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].

A caching scheme called Graceful Caching [5], in which a variable amount of memory is assigned to each image by using hierarchical coding format, has been proposed. This hierarchical image caching system has been shown to achieve substantial performance improvement [6].

As the cache capacity is limited, an important research issue is how to efficiently allocate cache memory. This so-called caching algorithm has attracted much research. However, most of the current available algorithms are for the conventional (hard) caching schemes. How to determine which images and which layers in the images should be cached and replaced is not mentioned. Also, most of the common algorithms merely concern themselves with how to remove the object from the cache. They show little consideration for how to fetch the object from the server to the cache.

In this paper, we therefore present an integrated caching algorithm for this Graceful Caching (soft cache) system.

We firstly carry out a theoretical analysis of the web access of the Graceful Caching. Based on this analytical result, we then propose an integrated caching algorithm in which both replacing and pre-fetching algorithms are considered. In our algorithm the object causing the low miss hit ratio will be removed from the cache and the one whose request probability is high will be pre-fetched from the server. We show that the necessary parameters in our proposed algorithm can be obtained from the information readily available in the proxy cache. Through simulations, we check the performance of our proposal when the related parameters are changed, and find that our proposal can efficiently improve the hit ratio and user response time against the previous algorithms.

This paper is organized as follows. In Sect. 2, we give an overview of the Graceful Caching. In Sect. 3, we introduce works related to the caching algorithm. In Sect. 4, we analyze web access of the Graceful caching and obtain the related results. Section 5 presents our

proposed algorithm. Simulation results are given in Sect. 6 and conclusions are shown in Sect. 7.

2. Overview of the Graceful Caching

Proxy caching has been one of the most important means of distributing web contents from servers to end clients. By keeping local copies of documents requested by clients and returning them locally to further requests for the same documents, proxy caching can reduce the amount of traffic and user response time.

Web proxy workload characteristics show that a large percentage of network traffic is caused by image documents. So it is important to efficiently cache image data.

Conventional cache management can be considered as "hard" caching because the object is either cached or not cached irrespective of data size.

To overcome this problem, we proposed a soft and flexible caching method, called Graceful Caching. In the Graceful Caching, we assume that a progressive image format is used and a low resolution version of the original image can be kept in the cache. By doing this, variable amounts of cache memory can be assigned to each image according to clients' demand.

The key idea of the Graceful Caching is firstly proposed for image database systems [5], and then it is expanded to a proxy cache system [6], [13].

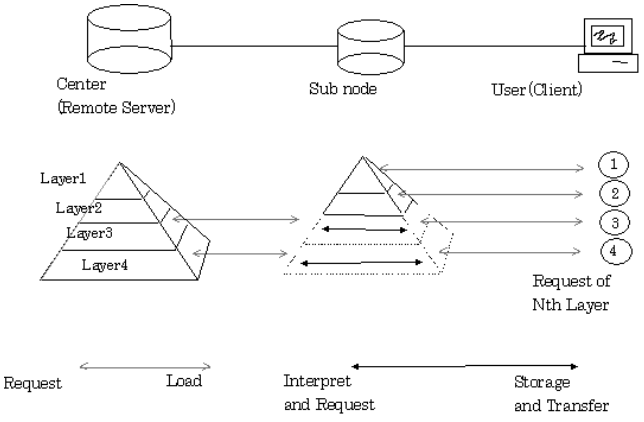


Fig. 1 Image database system with the graceful caching.

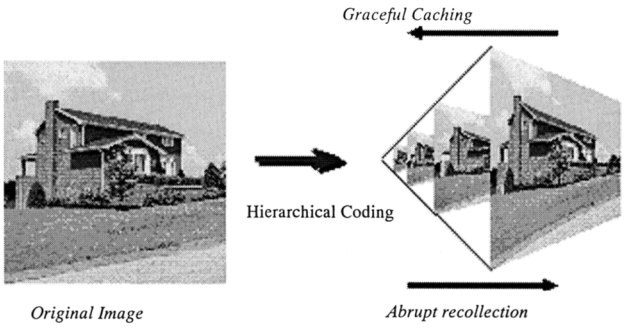


Fig. 2 Pyramidal image structure, oblivion, recollection.

Figure 1 illustrates a simple architecture of the Graceful Caching. Clients request image data from one of the center databases (server) via a sub-node (proxy cache). If the sub-database cached the required data, clients don't need to access the center server directly. The Graceful Caching utilizes hierarchical property of images to efficiently use the memory in the sub-node. Figure 2 shows an example of a graceful oblivion and recollection process in the image of pyramidal structure. The size of image cached in the sub-node is reduced each time when a given period of time for this data has elapsed (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).

Previous researches [6], [7], [13] proved that the Graceful Caching could provide substantial performance improvement compared with conventional caching schemes, i.e. hard caching. The advantage of this approach is that, in many cases, users will not need to download full resolution images from the server if an acceptable quality of image is retrieved. Furthermore, more images can be kept in the cache for user's re-utilization by using hierarchical image format.

3. Related Work to Caching Algorithms

How to efficiently allocate the cache memory to achieve optimal performance is, however, still an issue to be resolved. Because cache memory is limited compared with the capacity of the server, only a small part of all objects in the server can be kept in the cache. How to assign cache memory to different images according to clients' demands plays a very important role in cache performance [9].

The most commonly used algorithm, LRU (Least Recently Used), removes the least recently used one among the objects present in the cache. However, researches showed that one of the main weaknesses of this LRU is that the cache would be flooded by objects referenced only once, flushing out documents with higher probability of being reused.

In a related study, an algorithm called OPT-SOFT has been proposed especially for the Graceful Caching [8]. In this algorithm, access probability of every layer of every image is calculated according to its past access frequency. The layer of one image whose access probability is the lowest will be removed from the cache. One weakness of this algorithm is that it needs to calculate access frequency of all images, which will cause a huge computation load. On the other hand, since the access pattern of clients is dynamic, access probability that is calculated from the past record can't reflect current access behavior of clients very well. For example, objects with high reference counts that were popular before may not be necessarily popular now. Just like

a LFU (Least Frequency Used) algorithm, it prevents "dead" documents with high reference counts from being purged.

An algorithm called TTL-LRU has also been proposed [13]. In this algorithm, both access frequency and access interval are considered to assign an index to each layer of every image. In the TTL-LRU algorithm, the cache firstly compares the indices of the lowest layer (biggest size layer) of every object, then, removes the layer that has the smallest index value. However, a parameter α in the index, which significantly affects system performance, is not easily determined.

On the other hand, all of the above algorithms only concern how to remove objects from the cache. We call them "simple replacing" algorithms because they show little consideration of how to fetch objects from the server into the cache. Several recent studies suggest that prefetching techniques could be employed to further improve the cache performance, i.e. to anticipate and prefetch future client requests in an efficient manner [10], [16]–[19].

In this paper, we propose an integrated caching algorithm for the Graceful Caching system in which both replacing and pre-fetching will be considered based on theoretical analysis of the web access of the Graceful Caching.

4. Theoretical Analysis

This section is divided into two parts: in the first part, we will analyze web request distribution in the Graceful Caching and calculate theoretical request probability. In the second part, we will carry out a theoretical analysis of the access manner of the Graceful Caching and derive the relation between the consecutive requests for the same image.

4.1 Web Request Probability in the Graceful Caching

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

A=$\{a_{i,j}$; the j-th layer of image i $\}$,
i=$\{1, \cdots, N\}$, j=$\{1, \cdots, L$ $\}$,
B=$\{b_{i,j} = \{a_{i,1}$,$a_{i,2}, \cdots, a_{i,j}\}\}$,
P:$B \rightarrow [0, 1]$

Here, B is a Borel field, the family of partial subsets of A which is strictly regulated by Progressive JPEG encoding format [14]. 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. P is the probability measure that maps B into a $[0, 1]$ value, $P(b = b_{i,j}) = P(i, j)$.

We can obtain a web request probability for the j-th layer of image i by the following equation:

$$P(b_{i,j}) = \prod_{m=0}^{j-1} d_{k,m} P(b_{i,1}) = \frac{\Omega \prod_{m=0}^{j-1} d_{k,m}}{r_i^{\alpha}} \quad (1)$$

where the proof of this equation is given in Appendix A. Here, Ω, α are parameters of Zipf distribution, and r_i is the ranking of the image i. $d_{k,j}$ means a probability that a user will request the next layer ($(j+1)$-th) after getting the j-th layer. Note that $j = 0$ means that a user has no layer of the requested image yet, and we can know that $d_{k,0} = 1$ (if a user has no layer of the image he will request the first layer when he begins to request this image). k means the number of layers in the cache; when a user begins to request an image, there are k layers of that image stored in the cache at that time. We will explain $d_{k,j}$ in detail in the next part.

4.2 Analysis of $d_{i,j}$

Firstly, the access mechanism of the Graceful Caching will be introduced. When a user requests an image, all of the available layers stored in the cache will be automatically sent to the user in a gradual manner.

If the user is not satisfied with the quality of the received layers stored in the cache, he can click the 'reload' button. All of the rest data of the original image will then be loaded from the server.

Secondly, we will make an analysis about $d_{k,j}$. We define $D = [d_{k,j}]$ as the Greedy Degree of the j-th layer, which means a probability that a user requests the next layer ($(j+1)$-th) after getting the j-th layer, when there are k layers of image i stored in the cache. Since $d_{k,L}$=0 (when a user has already got all of the L layers, he will not request more layers), we can know k is a variable from 0 to L and j is a variable from 0 to $L-1$.

After this analysis, we can obtain the Greedy Degree by the following equation:

$$d_{k,j} = \begin{cases} p_k & (j = k) \\ 1 & (j \neq k) \end{cases} \quad (2)$$
$$k = \{0, \cdots, L\}, j = \{0, \cdots, L-1\}$$

where the detailed analysis of Eq. (2) is provided in Appendix B.

In Appendix B, we also prove that, if the original image is divided into 4 layers ($L = 4$) as a simple scenario, a following matrix can be defined.

$$D = [d_{k,j}] = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \end{bmatrix} = \begin{bmatrix} p_1 & 1 & 1 \\ 1 & p_2 & 1 \\ 1 & 1 & p_3 \end{bmatrix} \quad (3)$$
$$k = \{1, 2, 3\}, j = \{1, 2, 3\}$$

where $d_{4,j}$=1, $d_{0,j}$=1, and $d_{k,0}$=1.

An analysis of D_1 will be shown as an example: If there is only $k = 1$ layer in the cache and a user

has already got it, the probability that he will not be satisfied with the current quality and will request more layers is given by p_1. If he requests more layers, all of the remaining layers will be sent to him. He will get the data layer by layer. So the probability that he gets the third (fourth) layer after having got the second (third) layer is 1. That is to say: $D_1 = [p_1, 1, 1]$.

From Eq. (3), we can find that we only need to calculate three parameters (p_1, p_2, p_3) if images are divided into four layers in total.

5. Proposed Algorithm

In this section, the proposed integrated caching algorithm will be introduced. Our proposed algorithm consists of two parts. One is a replacing algorithm presented in Sect. 5.1. Another is a pre-fetching algorithm presented in Sect. 5.3. How to get the related parameters from information local to the cache will be introduced in Sect. 5.2. An issue about how to predict the related parameters will be discussed in Sect. 5.4.

5.1 Replacing Algorithm

Before we decide to remove a certain layer of one image to make room for a new coming object, we will calculate the request probability of this layer. If its request probability is high, which means a miss hit will be likely to happen after this layer is removed, we will not purge this layer from the cache.

Assume that k layers of image i are in the cache now, if we select the k-th of image i to be removed, we can know that the total number of layers of image i in the cache will be $k-1$. So the Greedy Degree in Eq. (1) should be $d_{k-1,m}$.

From Eq. (1) we can obtain the probability that a client requests the removed layer (k-th):

$$P(i,k) = \frac{\Omega \cdot \prod_{m=0}^{k-1} d_{k-1,m}}{r_i^{\alpha}} \tag{4}$$

From Eq. (2) we know:

$$d_{k-1,m} = \begin{cases} p_{k-1} & (m = k-1) \\ 1 & (m \neq k-1) \end{cases} \tag{2-A}$$

By substituting Eq. (2-A) into Eq. (4), we obtain,

$$P(i,k) = \frac{\Omega \cdot p_{k-1}}{r_i^{\alpha}} \tag{5}$$

As a result, the question is simplified to calculate the index as the Eq. (5) for the objects in the cache. The data whose index is the lowest (the probability of miss hit is the lowest) will be first to be removed from the cache.

5.2 How to Get the Necessary Parameters from Local Cache Information

In Eq. (5), Ω, α are parameters of Zipf distribution. As a result, as long as we get p_{k-1} and r_i (image ranking), the probability that a miss hit will happen if we remove one object can be calculated.

Here we introduce a way to calculate p_{k-1}, which reflects the client's Greedy Degree, directly from the local cache information.

We assume that $t\_send(k)$ is the times that a cache sends the top k layers to a user when there are only k layers currently cached in the cache. $t\_miss(k)$ is the times that a user goes on to request more layers after receiving the top k layers from the cache. Then, we can get:

$$d_{k,j} = \begin{cases} p_k = \frac{t\_miss(k)}{t\_send(k)} & (k = j) \\ 1 & (k \neq j) \end{cases} \tag{6}$$

As the cache itself has to calculate the hit ratio in order to evaluate cache performance, this method can directly achieve p_k from the local cache information (miss hit ratio) when cache sends all stored layers ($1 \sim k$) to the user.

r_i in Eq. (5) could be and is generally calculated according to the access record (reference counts) in a past period. As the access pattern is dynamic, objects with large reference counts in the past are not always as popular as before. For example, if an image was requested many times in the past, it would be kept in the cache for a long time because of its high index value no matter whether it is still popular or not. This kind of object is called a 'dead' file in the LFU (Least Frequently Used) algorithm and is believed to waste cache capacity.

Therefore, to grasp the current popularity of the objects, we apply an auto-regressive (AR) model to predict future access counts of the images from their past records. The goal is to estimate $Y_i(t+\tau)$ from image i's measured access history $\{Y_i(s)|s \in (t - q\tau, t)\}$, if given τ as the prediction interval. q is the order, i.e. the number of previous records to be used for prediction. Parameters ϕ_i and ∂ are determined by the AR model, respectively. Rankings of the images are determined by sorting the image in descending order with respect to the predicted access counts calculated by Eq. (7). For example, the ranking of the image with the maximum value of the predicted access counts takes 1.

$$\overset{\Lambda}{Y}_i(t+\tau) = \sum_{j=0}^{q} \phi_i Y_i(t - j\tau) + \partial \tag{7}$$

5.3 Pre-Fetching Algorithm

It has been proved that most web requests to a server

are for a very small set of objects [12], [15]. Recent studies suggest that prefetching those objects can further increase cache hit ratio as long as future client requests are tactfully anticipated. As the *AR* model is applied into our system, when a prediction interval is over we can consider the contents in the cache could also be adjusted at the same time. Here we will compare the access probability of the objects in the cache and objects in the server. If an object in the server, which has not been cached yet, has a higher probability than some object in the cache, it will be pre-fetched to the cache.

For images i with the predicted ranking r_i, we assume there are k layers in the cache now. Based on Eq. (2), we can derive:

$$d_{k,j} = \begin{cases} p_k & (k = j) \\ 1 & (0 \le j \le k-1 \cup k+1 \le j \le L-1) \end{cases}$$

There are two kinds of access probability for this image. One is for the cached data:

$$P(i,j) = \frac{\Omega \cdot \prod_{m=0}^{j-1} d_{k,m}}{r_i^{\alpha}} = \frac{\Omega}{r_i^{\alpha}} (1 \le j \le k) \quad (8)$$

Another is for the non-cached data still in the server.

$$P(i,j) = \frac{\Omega \cdot \prod_{m=0}^{j-1} d_{k,m}}{r_i^{\alpha}} = \frac{\Omega \cdot p_k}{r_i^{\alpha}} (k+1 \le j \le L) \quad (9)$$

The cache system can then compare the access probability of all parts of all images. An access probability threshold can be decided according to the cache capacity. If the access probability of one object in the cache is lower than this threshold, it will be removed from the cache. On the other hand, if the access probability of one object in the server is higher than this threshold, it will be pre-fetched from the server.

5.4 How Many Resources Should be Predicted

As there are numerous images in the server, it is unrealistic to predict all images' access frequency. Moreover, since references to images accessed only once account for a large fraction, it is not necessary to predict these once-accessed images in every prediction interval.

Previous researches showed that the distribution of web requests from a fixed group of users follows a zipf distribution. It has been proved that most web requests to a server are for a very small set of objects, for example top 10% [10]. Because of this property, it is enough to predict the aforementioned set of images.

We assume that image i's access frequency in last prediction period is $Y_i(\tau)$,

- If $Y_i(\tau) > Threshold$, an *AR* model is applied to predict its future access in the coming period.
- If $Y_i(\tau) \le Threshold$, the object's access count is counted according to the LFU algorithm, in which the image's index is counted and increased by real access within this period, instead of predicting its future access frequency.

Threshold is a value of the image of which ranking is Top 10 or others.

One obvious question is whether there is a case that an image that was not popular before becomes very popular within the current period? Since this image was not in the popular set of images in the last period, there is no prediction value for it. Could the system efficiently handle this situation?

Here, we will give an explanation. As this kind of image's index is counted according to the LFU, its index will be increased by unity after every access. If it is accessed many times in one period, its index will become high and it will be retained in the cache for a long time for re-utilization. Moreover, because of the frequent access in this period, this image could be added into the popular set (prediction set) when the next prediction period begins.

Overall, we think that our proposal is an integrated one in which both pre-fetching and replacing algorithms could be carried out. The pre-fetching algorithm focuses on how to fetch contents from the server into the cache. The replacing algorithm emphasizes how to replace contents in the cache to make room for new coming objects. The related parameters of the algorithms can be obtained on the basis of statistical data collected from the cache.

6. Evaluation of Algorithms

6.1 Simulation Conditions

We assume that there are 5000 different images in the server. Images are supposed to be accessed according to Zipf frequency distributions [11], [12]. The Zipf parameter α is 0.8.

Here we set the threshold of prediction to 20%, which means the Top 20 most popular images' access frequency will be predicted.

The original images will be encoded into a hierarchical image format with four layers. The picture size of different layers is proportional to 5:13:22:59 according to the Progressive JPEG format, in which the fourth layer has the largest size. Cache size is 5% of the total size of all images in the server.

Since the original image is divided into 4 layers, from Eq. (3) we find that we only need to calculate three parameters (p_1, p_2, p_3) to get the information of Greedy Degree. As for the values of p_{k} (k=1,2,3), they are firstly evaluated by using a group of real data as shown in [20]. The results for the more general case, where (p_{k}) is variable from 0.0 to 1.0, will be reported later in

the simulation section. We then experimentally compare the performance of four caching algorithms: LRU, OPT-SOFT, TTLLRU and the proposed algorithm as follows.

6.2 Simulation Results

As we have discussed in Sect. 5.1, the index of our algorithm in Eq. (5) is mainly determined by two parameters, p_{k}, and r_{i}. In this section, simulation results will be given. We will evaluate the performance of our proposal with respect to these parameters. The first simulation will show the system performance related to r_{i} (access patterns). The second simulation will show the system performance related to p_{k} (Greedy Degree).

6.2.1 Simulation: Effect of Access Patterns

In this section we analyze the performance of the algorithm with different access patterns (r_{i}). In this simulation, two extreme access situations are evaluated.

In one case, an access pattern is static, where the image ranking is not changed at all during the simulation.

In the other case, an access pattern is intensively changed temporarily in order to investigate the robustness of every algorithm under dynamic access pattern changes. We assume that the simulation time is 30 days and the ranking of all images is changed about 5% within every 8 days.

Figure 3 shows the hit ratio when different algorithms are used under the static access pattern. As the proposed algorithm consists of a pair of algorithms, the replacing algorithm and pre-fetching algorithm, the replacing algorithm is firstly carried out and we find that it achieves higher hit ratio than any of the other conventional algorithms in Fig. 3. Then, we go on to carry out the pre-fetching algorithm. Its result is shown as Proposal (Replacing + Pre-fetching) and the hit ratio proved to be further increased. Not only OPT-SOFT but also TTL-LRU and the proposed algorithm outperform the commonly used algorithm, LRU. OPT-SOFT and TTL-LRU provide a modest improvement while the proposed algorithm achieves the best result.

Next, we also evaluate their performances when the clients' access pattern is intensively changed. From Fig. 4 we can see that the proposed algorithm is much better than the others, although its performance drops slightly compared with the result under the static access pattern. However, OPT-SOFT drops most among the four algorithms. The reason is that the access frequency in the last period is used to decide images' priorities. Furthermore, it can not reflect the current image popularity. Although LRU showed poor results in Fig. 3, its performance is very stable under the dynamic access pattern. This is because it always keeps the most recently used image in the cache. Performance of TTL-LRU is close to LRU as shown in Fig. 4. The proposed algorithm works well under the dynamic access pattern, too. This is because, on one hand, prediction of popular images can prevent the replacing algorithm from removing current popular objects from the cache. On the other hand, pre-fetching can also help the cache to keep user's favorite images in the cache. In Fig. 4 the replacing algorithm can increase the hit ratio about 5.4% compared with LRU. By carrying out the pre-fetching algorithm as the second step, the hit ratio can be further increased about 3.8%.

From Fig. 3 and Fig. 4, we can find that our proposed algorithm is the best and the most adaptive one since it achieves the best results under both of the two extreme access patterns.

Figure 5 and Fig. 6 compare relative response time with respect to the commonly used algorithm, LRU. After considering the development of recent network access, we first assume that the bandwidth between clients and a proxy is 100 Mbps and the bandwidth between servers and a proxy is 10 Mbps. Figure 5 shows the improvements after carrying out the replacing algorithm and prefetching algorithm in succesion. It can be found that the proposed algorithm including replacing and pre-fetching has the shortest relative response time under a static access pattern. It can reduce the response time up to 19%, which is almost 6% more than the reduction that OPT-SOFT has achieved.

Figure 6 shows the results when the clients' ac-

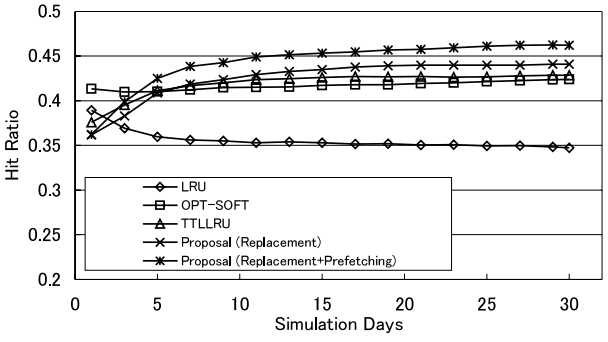


Fig. 3 Hit ratio under static access pattern.

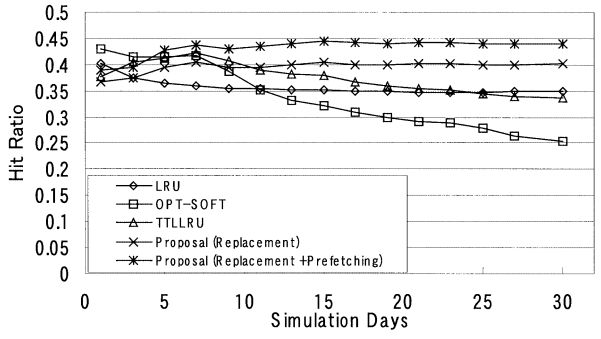


Fig. 4 Hit ratio under dynamic access pattern.

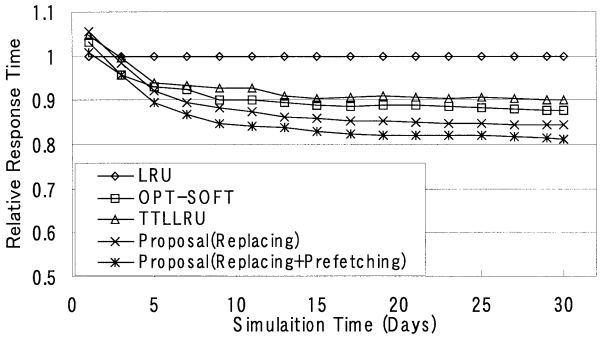


Fig. 5 Relative response time under static access pattern.

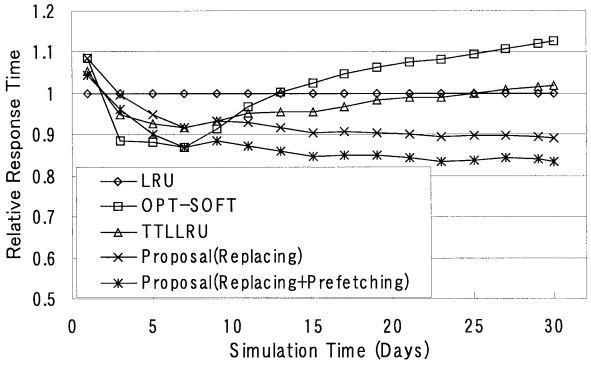


Fig. 6 Relative response time under dynamic access pattern.

cess pattern is intensively changed. The replacing algorithm can reduce response time about 11% compared with LRU. After carrying out both replacing and prefetching algorithm the response time can be reduced up to 18%. Compared with the results in Fig. 5, our proposed algorithm is the most stable one since its performance under the different access patterns doesn't change very much either. Note that the other algorithms' performances drop drastically when the popularity ranking of images is changed. The results in Fig. 3–Fig. 6 also verify that the algorithm that gets a high hit ratio also obtains a shorter response time.

In Sect. 5.4 we have discussed that it is not necessary to predict the access counts for all images because web access web follows a Zipf distribution. In the simulation in Fig. 3–Fig. 6, only top 20% popular images' access frequency was predicted.

In Fig. 7 we will test the performance of our integrated proposal including both replacing and prefetching when the predicted percentage is changed from 0.0 to 1.0. Form Fig. 7 we can find the hit ratio is relatively low when the predicted percentage is lower than 10%. And the performance becomes stable after predicting top 20% popular images.

The reason can be found from web access distribution. According to Zipf distribution, most of the requests are for a very small set of objects. Top 20% of the images account for about 70% of all requests seen by the cache, and top 50% of the images account for

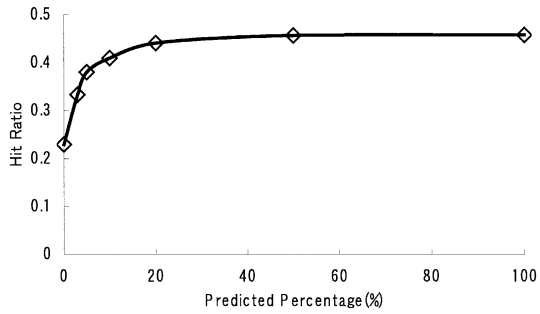


Fig. 7 Performance of the proposal vs. predicted percentage.

85% of all requests [12]. When more than top 20% images' access counts are predicted, the cache can know that most of all requests are to the cached images with high probability. That is to say: the access pattern is almost predicted. As a result, the performance becomes stable. The result in Fig. 7 proves the conclusion that only predicting part of popular images' access count is enough for predicting the access pattern of web access.

6.2.2 Simulation: Effect of Greedy Degree

In this section we compare performances of the algorithms with respect to p_{k}. In Sect. 6.2.1, user's Greedy Degree was simulated according to the data investigated by [20]. Here we show the effects of variables (p_{k}) which is changed from 0.0 to 1.0. As the original images are encoded into a hierarchical format with four layers, p_{k} (k=1,2,3), will be tested. As for the other simulation conditions, they are the same as in Sect. 6.2.1.

Since LRU and TTL-LRU got better result in Fig. 4 and their performances are very close, we will only compare performances between our integrated proposal including both replacing and pre-fetching and the LRU algorithm in order to show the proposed one's improvement over the conventional method.

We firstly test the situation when p_2 and p_3 are variables. From Fig. 8 we can know the relative improvement over LRU is from 1.2 to 1.4, which means that the proposed one achieves much improvement in cache performance.

Also notice that the improvement is better when p_3 is small. The reason is that, under this situation, the proposed algorithm intends to remove the big size layer first from the cache because it knows that users don't always request more layers of one image. Therefore, more cache room can be provided for the new coming objects. However, since the LRU only removes the least recently accessed object irrespective of its size, its performance is not affected by p_{k}. So, the relative hit ratio under this situation becomes higher in the proposed algorithm. Although the performance with high p_3 value is not as good as before, we find that the proposed algorithm still gives good performance (around

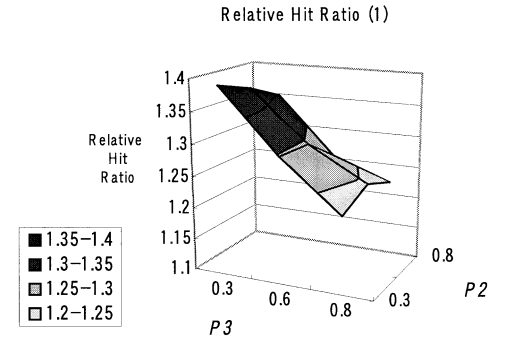


Fig. 8 Relative hit ratio when p_2 and p_3 are variable.

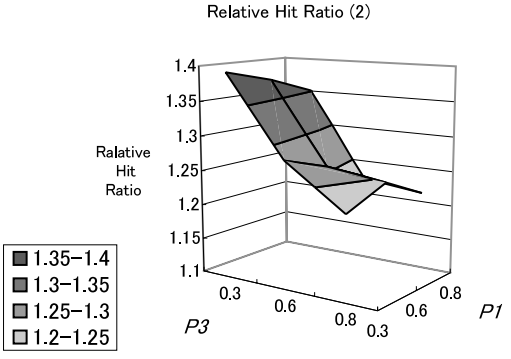


Fig. 9 Relative hit ratio when p_1 and p_3 are variable.

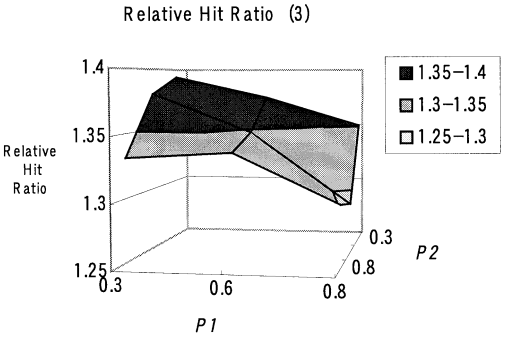


Fig. 10 Relative hit ratio when p_1 and p_2 are variable.

120%).

Figure 9 is the relative hit ratio when p_2 is fixed. The result is similar to Fig. 8. The relative improvement over the LRU stays in an area from 1.2 to 1.4.

Figure 10 is the relative hit ratio when p_3 is fixed. The result in Fig. 10 is flatter than the above two cases. When p_1 (p_2) changes it means that the probabilities that a user requests the second layer (third layer) is changed. Because the data sizes of the second layer and the third layer are not as large as the last layer, varying parameters don't affect system performance to the same extent as before.

Overall, we can conclude that our proposed algorithm has the best performance among the four algorithms investigated no mater how the relative parameters change.

7. Conclusion

In this paper, we provided a theoretical analysis of the web access behavior of the Graceful Caching. Based on the analysis, we proposed an integrated caching algorithm in which both cache replacing and image prefetching were carried out according to a mathematical scheme. We also showed the way to obtain the necessary parameters for the proposed algorithm from the readily available information in the cache. We quantitatively evaluated our proposal when the related parameters were changed, and compared our proposal with other existing ones. The results showed that the proposed algorithm outperforms all of the previous ones. Both cache hit ratio and user response time have been efficiently improved.

As an increasing number of streaming media objects are being distributed over the Internet, caching and distributing streaming media in Content Delivery Networks by using layered encoded method will be considered as an extension of our work in the future.

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.COM-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, vol.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, Aug. 1996.

[6] K. Kamogawa, D. Nakajima, and Y. Yasuda, "Proxy server systems with hierarchical image caching," J. Inst. Image Electron. Eng. Jpn., 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," Proc. IEEE Signal Society Workshop on Multimedia Signal Processing, Princeton, NJ, June 1997.

[8] J. Kangasharju, Y. Kwon, A. Ortega, X. Yang, and K. Ramchandran, "Implementation of optimized cache replenishment algorithms in a soft caching system," Proc. IEEE Signal Society Workshop on Multimedia Signal Processing, Redondo Beach, CA, Dec. 1998.

[9] M. Arlitt, R. Friedrich, and T. Jin, "Performance evaluation of web proxy cache replacing 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," Proc. INET' 98, Geneva, Switzerland, July 1998.

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

issues," Proc. 3rd International WWW Caching Workshop, Manchester, UK, June 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, New York, NY, USA, March 1999.
[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," J. Inst. Image Electron. Eng. Jpn., vol.28, no.5, pp.605–611, Oct. 1999.
[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," Proc. ACM SIGMETRICS Atlanta, pp.178–187, Georgia, USA, 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," Proc. USENIX Symposium on Internet Technology and System, Monterey, California, USA, Dec. 1997.
[18] Z. Su, T. Washizawa, J. Katto, and Y. Yasuda, "A new prefetching algorithm for Graceful Caching system," 2001 General Conference of IEICE, March 2001.
[19] Z. Su, T. Washizawa, J. Katto, and Y. Yasuda, "Performance improvement of Graceful Caching by using request frequency based pre-fetching algorithm," IEEE TENCON, Singapore, Aug. 2001.
[20] K. Saito, Performance improvement of caching by using human's access degree for images, Graduate Thesis, Waseda University, March 2001.
[21] A. Beck and M. Hofmann, "Enabling the Internet to delivery content-oriented services," The 6th International Web Caching and Content Distribution, Boston, USA, June 2001.
[22] J. Kangasharju and K.W. Ross, "Performance evaluation of redirection schemes in content distribution networks," The 5th International Web Caching and Content Delivery Workshop, Lisbon, Portugal, May 2000.
[23] R.P. Doyle, J.S. Chase, S. Gadde, and A.M. Vahdat, "The trickle-down effect: Web caching and server request distribution," The 6th International Web Caching and Content Delivery Workshop, Boston, June 2001.

Appendix A: Web Access Analysis

Here we assumed the first layer has the smallest size and the lowest resolution. $P_{i,j}$ is the probability that a user requests for the j-th layer of image i.

We define a parameter, called Greedy Degree ($d_{k,j}$), which means the probability that a user requests for the next layer ($(j+1)$-th) after retrieving the j-th layer. k means that, when a user begins to request an image, there are k layers of that image stored in the cache. Using these parameters, we can get the conditional probability $b_{i,j+1}$ given $b_{i,j}$: $P(b_{i,j+1}\,|b_{i,j}\,) = d_{k,j}$.

According to the decoding constraint of the layered encoded image, the $(j+1)$-th layer can't be provided to the user before he has already got the j-th layer. That is:

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

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

$$\begin{aligned}&P(b_{i,j+1}) = P(i,j+1)\\&= P(b_{i,j+1}\,|b_{i,j}\,)\times P(b_{i,j})\\&+ P(\,b_{i,j+1}|\,(B\setminus\{b_{i,j}\}))\times P(B\setminus\{b_{i,j}\})\\&= d_{k,j}\times P(b_{i,j}) + 0\times P(B\setminus\{b_{i,j}\})\\&= d_{k,j}\times P(b_{i,j}) = d_{k,j}\times P(i,j)\end{aligned}$$

Since $P(b_{i,j+1}) = d_{k,j}\cdot P(b_{i,j})$, we can get:

$$P(b_{i,j}) = \prod_{m=0}^{j-1} d_{k,m}\cdot P(b_{i,1})$$

According to a Zip'f distribution, the probability that the image i is requested is as follows:

$$P(b_{i,1}) = \frac{\Omega}{r_i^{\alpha}}$$

where Ω, α are parameters of Zipf distribution, and r_i is the ranking of the image i. We can get the probability that a request happens for the j-th layer of image i by

$$P(b_{i,j}) = \prod_{m=0}^{j-1} d_{k,m}P(b_{i,1}) = \frac{\Omega\prod_{m=0}^{j-1} d_{k,m}}{r_i^{\alpha}}$$

Appendix B: Analysis of $d_{i,j}$

We define $D = [d_{k,j}]$ as the Greedy Degree of the j-th layer, which means the probability that a user requests the $(j+1)$-th layer after getting the j-th layer when there are k layers of image i stored in the cache. Since $d_{k,L} = 0$ (means that, because a user has already got all of the L layers, he will not request more layers), we can get $k = \{0,\cdots,L\}$, $j = \{1,\cdots,L-1\}$.

We know $d_{L,j} = 1$ (since all of the layers are in the cache, at the time of $k = L$, a user can get all of the layers from the cache. This corresponds to the fact that a user keeps on requesting one layer after another.), $d_{0,j} = 1$ (since there is no layer of the image available, at the time of $k = 0$, a cache will fetch all of the layers from the server to the user, that makes the user get data layer by layer later.) and $d_{k,0} = 1$ (if a user has no layer of the image, he will request the first layer when he begins to request this image). As a result we are only interested in the situation in which k is a variable from 1 to $L-1$ and j is a variable from 1 to $L-1$. As for $D = [d_{k,j}]$ ($k = \{0,\cdots,L-1\}$, $j = \{1,\cdots,L-1\}$), we can calculate:

$$D = [d_{k,j}] = \begin{bmatrix} D_1 \\ D_2 \\ . \\ D_k \\ . \\ D_{L-1} \end{bmatrix}$$

$$= \begin{bmatrix} p_1 & 1 & . & . & . & 1 \\ 1 & p_2 & 1 & . & . & . \\ 1 & 1 & . & 1 & . & . \\ . & . & 1 & p_k & 1 & . \\ . & . & . & 1 & . & 1 \\ 1 & . & . & . & 1 & p_{L-1} \end{bmatrix}$$

Proof:

We assume that there are k layers of the image stored in the cache when a user begins to send a request for that image. Since its k layers are available in the cache, all of the data from the first layer to the k-th layer will be sent automatically as soon as he requests the image. As a result, the user can get those layers no matter whether he designedly requests these data or not. So we can conclude: $d_{k,j}$=1($j = [1, \cdots, k-1]$). If the user is not satisfied with the current quality, the probability that he wants to reload more is $d_{k,k}$= p_{k}. ($j = k$). Since the rest of data from the $(k+1)$-layer to the last layer will be reloaded and sent to the user at the same time, we can get $d_{k,j}$=1($j = [k+1, \cdots, L-1]$).

Finally, we obtain the greedy degree by

$$d_{k,j} = \begin{cases} p_k & (j = k) \\ 1 & (j \neq k) \end{cases}$$

$$k = \{1, \cdots, L-1\}, j = \{1, \cdots, L-1\}$$

where $d_{L,j} = 1$ ($k = L$), $d_{0,j} = 1$ ($k = 0$) and $d_{k,0} = 1$ ($j = 0$), respectively. That is to say:

$$d_{k,j} = \begin{cases} p_k & (j = k) \\ 1 & (j \neq k) \end{cases}$$

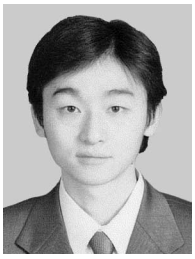
$$k = \{0, \cdots, L\}, j = \{0, \cdots, L-1\}$$

In a simple scenario, if the original image is divided into 4 layers (L=4), from the above equation, D is given by

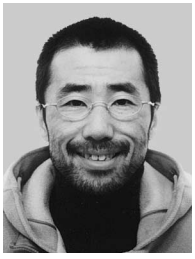
$$D = [d_{k,j}] = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \end{bmatrix} = \begin{bmatrix} p_1 & 1 & 1 \\ 1 & p_2 & 1 \\ 1 & 1 & p_3 \end{bmatrix}$$

$$k = \{1, 2, 3\}, j = \{1, 2, 3\}$$

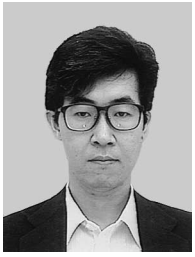
where $d_{4,j} = 1$, $d_{0,j} = 1$ and $d_{k,0} = 1$.



Zhou Su is currently working toward his Ph.D. Degree at the Graduate School of Science and Engineering, Waseda University, after receiving his BE and ME degree from Xi'an Jiaotong University, Xi'an, China in 1997 and 2000, respectively. He was an exchange student between Waseda and Xi'an Jiaotong University from 1999 to 2000. From 2001 he has been a research associate at Waseda University. His research interest is in the field of multimedia multi-media communication, web performance and network traffic. He received the SIEMENS Prize in 1998, and received Rockwell Automation Master of Science Award in 1999. He is a student member of the IEEE and IEE.



Teruyoshi Washizawa is currently an associate scientist at Canon Research Center, after receiving his B.S. and M.S. degree from Hokkaido University, in 1982 and 1984,respectively. From 1998 to 2000, he was a visiting researcher at Waseda University. His research interests include neural networks, statistical learning, image processing, and database retrieval. He received the Hatakeyama Awards from JSME in 1982. He is a member of JNNS, IEEE, and IIEEJ.



Jiro Katto was born in Tokyo, Japan, in 1964. He received the B.S., M.E. and Ph.D. degrees in electrical engineering from University of Tokyo in 1987, 1989 and 1992, respectively. He worked for NEC Corporation from 1992 to 1999. He was also a visiting scholar at Princeton University, NJ, USA, from 1996 to 1997. He then joined Waseda University in 1999, where he has been an associate professor at the Department of Electronics, Information and Communication Engineering, School of Science and Engineering. His research interest is in the field of multimedia signal processing and multimedia communication systems such as the Internet and mobile networks. He received the Best Student Paper Award at SPIE's conference of Visual Communication and Image Processing in 1991, and received the Young Investigator Award of IEICE in 1995. He is a member of the IEEE and the IPSJ.

Yasuhiko Yasuda was born in Tokyo on July 7, 1935. He received B.E. and M.E. degrees in Electrical Engineering, and D.E. degree in Electronic Engineering from the University of Tokyo, in 1958, 1960 and 1963, respectively. In 1963 he joined the Institute of Industrial Science, the University of Tokyo as an assistant professor and was promoted to a full professor in 1977. Since retiring from the University of Tokyo in August of 1992, Dr. Yasuda has been professor at the Department of Electronics, Information and Communication Engineering, School of Science and Engineering, Waseda University. During his long career of research, he produced numerous useful achievements including invention of delta sigma modulation, development of 3-level VSB high speed transmission scheme for newspaper fax machines corresponding to the CCITT G2 fax standard, proposal of coding methods for digital facsimile images, proposal of hierarchical image coding and processing, etc. His current interest is in the fields of image coding and processing, mobile communications and satellite communications. He is a past President of Institute of Electronics, Information and Communication Engineers and a past President of the Institute of Image Electronics Engineers of Japan. He served acting chairman of Telecommunication Technology Council (MPT), a member of Broadcast Technology Council (NHK) and a member of board of directors, Engineering Academy of Japan. He is now serving Chairperson of Radio Regulatory Council. Dr. Yasuda has been awarded numerous prizes from various academic organizations including the IEICE Outstanding Achievement Award.