Exploring Aggregate Effect with Weighted Transcoding Graphs for Efficient Cache Replacement in Transcoding Proxies

Cheng-Yue Chang and Ming-Syan Chen
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC
E-mail: {cychang@arbor.ee.ntu.edu.tw; mschen@cc.ee.ntu.edu.tw}

Abstract

This paper explores the aggregate effect when caching multiple versions of the same Web object in the transcoding proxy. Explicitly, the aggregate profit from caching multiple versions of an object is not simply the sum of the profits from caching individual versions, but rather, depends on the transcoding relationships among them. Hence, to evaluate the profit from caching each version of an object efficiently, we devise the notion of a weighted transcoding graph and formulate a generalized profit function which explicitly considers the aggregate effect and several new emerging factors in the transcoding proxy. Based on the weighted transcoding graph and the generalized profit function, an innovative cache replacement algorithm for transcoding proxies is proposed in this paper. Experimental results show that the algorithm proposed consistently outperforms companion schemes in terms of the delay saving ratios and cache hit ratios.

1. Introduction

Recent technology advances in mobile communication have ushered in a new era of personal communication. Users can ubiquitously access the Internet via many mobile appliances, such as handheld PCs, personal digital assistants (PDAs), and WAP-enabled cellular phones. As these devices are divergent in size, weight, input/output capabilities, network connectivity and computing power, differentiated services should be tailored and delivered in a certain way to meet their diverse needs. In addition, users may have different content presentation preferences. Both lead to the demand of transcoding technologies to adapt the same Web object to various mobile appliances [6].

Transcoding is defined as a transformation that is used to convert a multimedia object from one form to another, frequently trading off object fidelity for size. For the mobile appliances featured with lower-bandwidth network connectivity, transcoding can be used to reduce the object size by lowering the image resolution or downscaling the image size. For the mobile appliances which only accept a text, transcoding can be used to covert the image or speech into a text. As pointed in [5], from the aspect of the place where transcoding is performed, the transcoding technologies can be classified into three categories, i.e., server-based, client-based, and proxy-based approaches. In the server-based approaches [12], Web objects are off-line transcoded to multiple versions and stored in the server disks. The advantage of this approach is that no additional delay will be incurred by transcoding during the time between the client issues a request and the server responses to it. The drawback is, however, keeping several versions of the same object in the server may cost too much storage space. Further, this approach is not flexible in dealing with the future change of clients' needs. Conversely, in the client-based approaches, transcoding is left for mobile clients for considerations. The advantage of this approach is that it can preserve the original semantic of system architecture and transport protocols. However, transcoding at the client side is extremely costly due to the limited connection bandwidth and computing power of a mobile device. For these reasons, it will be better to transcode the Web objects at the intermediate proxies. Many studies have recently been conducted to explore the advantages of proxy-based approaches [6][7][9][10] where an intermediate proxy is able to on-the-fly transcode the requested object to a proper version according to the client's specification before it sends this object to the client. Such an intermediate proxy which possesses the transcoding capability is referred to as a transcoding proxy in this paper.

While the transcoding proxy is attracting more and more attention, it is noted that a transcoding proxy, just as a tra-

ditional Web proxy, plays an important role in the functionality of caching. To enable the cache replacement algorithms devised for traditional Web proxies [1][2][3][14][17] to handle the situations in transcoding proxies, extensions to these traditional algorithms are needed. In [5], two extended strategies, which are called *coverage-based* and *demand-based* strategies, are proposed. Note that when transcoding the requested object to the specified version before sending this object to the client, the transcoding proxy has the opportunity of caching either the original version, the transcoded version, or both versions of the object in the local memory. In view of this, the *coverage-based* strategy in [5] is designed to choose the original version of the object to cache, whereas the *demand-based* strategy is designed to choose the transcoded one to cache. In addition, most cache replacement algorithms employ an *eviction function* to determine the priorities of objects to be replaced. The LRU (Least Recently Used) algorithm, for example, evicts the page with the largest elapsed time since the last access of that page. The LFU (Least Frequently Used) algorithm, on the other hand, evicts the page with the smallest reference rate in the cache. Collaborating the *eviction function* of the traditional cache replacement algorithm with these two strategies, we can decide the priority of the cached object to be evicted and the version of the new coming object to be cached. For example, the *demand-based* extension to LRU algorithm in [5] replaces the object of the largest elapsed time since its last access with the transcoded version of the new coming object. The *coverage-based* extension works similarly except the evicted object is replaced with the original version of the new coming object.

Note that both the *coverage-based* and *demand-based* extensions in [5] are designed for efficient Web caching. As a research work along this direction, we shall design in this paper a cache replacement algorithm for transcoding proxies that maximizes the performance of Web object caching. The motivation for our study is mainly due to the new emerging factors in the environment of transcoding proxies. First, transcoding incurs additional delays, and this factor should be included in the *eviction functions* for consideration whenever the eviction priority is determined. For example, suppose there are two cached objects, say, A and B, having the same eviction priority according to some eviction function which does not consider the transcoding delay. Clearly, if the transcoding delay of A is longer than that of B, A should be given a higher eviction priority than B so as to shorten the response time. Without considering the transcoding delays, the traditional cache replacement algorithms could make a wrong replacement decision. Second, different versions of the same object are of different

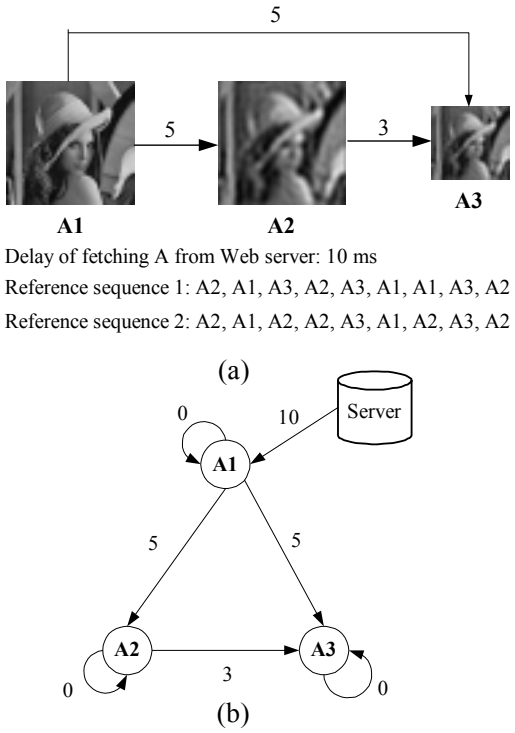

Figure 1. Illustrative examples.

sizes. Specifically, the transcoded version is usually of a smaller size than the original one. Thus, the strategy which chooses the transcoded version to cache may admit more objects with the same cache capacity. An *eviction function* is thus required to take this object size factor into consideration. Third, the reference rates to different versions of an object should be considered separately because the distribution of them could affect the caching decision. This can be seen by two contrary examples illustrated in Figure 1. Assume that *Lena image* has three versions (i.e., $A1$, $A2$ and $A3$). $A1$ is the original version with full resolution and size. $A1$ can be transcoded to $A2$ which is of the same size but a lower resolution. $A1$ can be also transcoded to $A3$ which is of a lower resolution and a smaller size. In addition, $A2$ can be transcoded to $A3$, whereas $A3$ is the least detailed version which cannot be transcoded any more. The transcoding delay for $A1$ to $A2$ and that for $A1$ to $A3$ are both assumed to be 5 ms, and the transcoding delay for $A2$ to $A3$ is 3 ms. The transmission delay of fetching object A from Web server is assumed to be 10 ms. Consider the reference sequence 1 in Figure 1, where the numbers of references to $A1$, $A2$ and $A3$ are all equal to 3. By caching $A1$, we can get the delay saving equal to 90 ms, which is obtained by $3 * 10 + 3 * (10 + 5 - 5) + 3 * (10 + 5 - 5)$. By caching

$A2$, we can, however, get the delay saving only equal to 81 ms, which is obtained by $3 * (10 + 5) + 3 * (10 + 5 - 3)$. As such, caching a more detailed version can get more benefit in this case. However, in reference sequence 2 in Figure 1, the numbers of references to $A1$, $A2$ and $A3$ are 2, 5 and 2 respectively. By caching $A1$, we can get the delay saving equal to 90, which is obtained by $2 * 10 + 5 * (10 + 5 - 5) + 2 * (10 + 5 - 5)$. In contrast, by caching $A2$, we can get the delay saving equal to 99, which is obtained by $5 * (10 + 5) + 2 * (10 + 5 - 3)$. In this case, caching a less detailed version will get more benefit than caching a more detailed one.

Also, consider the example in reference sequence 1 in Figure 1, the individual delay savings from caching $A1$ and $A2$ are 90 and 81 ms respectively. However, the aggregate delay saving from caching $A1$ and $A2$ at the same time is not simply the sum of the individual delay savings (i.e., $90 + 80 = 171$ ms), but rather, depends on the transcoding relationship among them. Note that the transcoding relationship in Figure 1(a) can be depicted by a directed graph with weights on edges as shown in Figure 1(b). Such a directed graph is called a *weighted transcoding graph*, which will be formally defined in Section 2. In the example of Figure 1, when caching $A1$ and $A2$ at the same time, the subsequent references to $A2$ and $A3$ are not necessary to be supported by the transcoding from $A1$ any more. Instead, they can be supported by the transcoding from $A2$ because there is no transcoding between $A2$'s, and the transcoding cost from $A2$ to $A3$ is smaller than that from $A1$ to $A3$. Hence, the delay saving from caching $A1$ is over-evaluated, and should be revised as $30 = 3 * 10$. The aggregate delay saving from caching $A1$ and $A2$ together is thus 111 rather than 171. Such an over-evaluation will probably result in a wrong cache replacement decision as will be explained below.

Object	Delay saving	Size
W	90	8
A1	90	10
X	85	8
Y	75	8

Replace Y with A2

Object	Delay saving	Size
W	90	8
A1, A2	111	18
X	85	8

Figure 2. Wrong cache replacement decision.

In the case of Figure 2, assume that the cache capacity is 34 Kbytes, and the cache is filled with W, $A1$, X and Y. The computed delay saving and size of each object is listed in the left table of Figure 2. Suppose here comes a new object $A2$. In the example of Figure 1, we have calculated the delay saving from caching it is 122. Without considering the aggregate effect of caching $A1$ and $A2$ together, the cache replacement algorithm will choose to replace Y with $A2$ because the delay saving from caching Y is smaller than that from caching $A2$. The resulting cache will, thus, become the one as listed in the right table of Figure 2. Recall that the aggregate delay saving from caching both $A1$ and $A2$ is 111 rather than 171. Thus, the overall delay saving of the entire cache decreases from 340 to 286 as Y is replaced with $A2$. Obviously, replacing Y with $A2$ is a wrong cache replacement decision.

Consequently, we propose an efficient cache replacement algorithm for transcoding proxies in this paper. Specifically, we formulate a *generalized profit function* to evaluate the profit from caching a version of an object. This *generalized profit function* explicitly considers several new emerging factors in the transcoding proxy and the aggregate effect of caching multiple versions of the same object. As explained, the aggregate effect is not simply the sum of the delay savings from caching individual versions of an object, but rather, depends on the transcoding relationship among these versions. Thus, the notion of a *weighted transcoding graph* is defined to represents the transcoding relationship among the different versions of an object. In addition, to evaluate the aggregate effect properly, we devise *Procedure MATC (standing for Minimal Aggregate Transcoding Cost)* to find the subgraph of the weighted transcoding graph. This subgraph shows the transcoding relationship which minimizes the aggregate transcoding cost when caching a certain set of versions of the object. Utilizing the *generalized profit function* as the eviction function, we propose in this paper an innovative cache replacement algorithm for transcoding proxies. This algorithm is referred to as algorithm *AE (standing for Aggregate Effect)* for the reason that it explores the aggregate effect of caching multiple versions of an object in the transcoding proxy. Using an event-driven simulation, we evaluate the performance of our algorithm under several circumstances. By varying the simulation parameters, we observe the performance impacts of two important parameters, including the cache capacity and γ_{FD} ratio. In the simulation, algorithm *AE* is compared with several companion schemes (i.e., *coverage-based* LRU, *demand-based* LRU, *coverage-based* LRUMIN and *demand-based* LRUMIN). The experimental results shows that algorithm *AE* proposed consistently outperforms the companion schemes in terms of the primary performance metric, delay saving ratio, and also the secondary performance metrics, hit ratios.

The rest of this paper is organized as follows. In Section 2, we introduce the system environment where the caching

issues in the transcoding proxy are considered. We propose our cache replacement algorithm for transcoding proxies in Section 3. In Section 4, we empirically evaluate the performance of several algorithms. We conclude this paper with Section 5.

2. System Environment

To facilitate our later discussion in this paper, we describe the system environment in this section. The system architecture is presented in Section 2.1. The Web object transcoding is described in Section 2.2.

2.1. System Architecture

This paper studies cache replacement algorithms for the transcoding proxy which typically interconnects two heterogeneous networks. A well-known example of such a transcoding proxy is the WAP proxy or a gateway, which interconnects the wireless network and the Internet. The mobile clients connect to the WAP proxy via the wireless network, whereas the WAP proxy connects to the Web servers via IP network. The mobile clients are capable of requesting and rendering WAP content. They vary widely in their features such as the screen size/color, the processing power, storage, and user interface. The clients can describe their capabilities via User Agent Profiles (UAProfs) [16] supported by WAP. UAProfs are initially conveyed when WSP sessions are established with the WAP proxy. Keeping these UAProfs in the proxy, the WAP proxy knows the preference of each client, and will on-the-fly transcode the requested object to a proper version before delivering to it. Moreover, as pointed out in [15], to speed up wireless data access, the WAP proxy also acts as the role of an caching proxy. In such a circumstance, the WAP proxy has the opportunity of caching either the original version, the transcoded version, or both versions of the object for future references. This emerging caching model justifies the problem we deal with in this paper. Note that while being applicable to the WAP architecture, the cache replacement algorithm we shall devise is for general transcoding proxies and by no means restricted to the WAP applications.

2.2. Web Object Transcoding

Transcoding is defined as a transformation that is used to convert a multimedia object from one form to another, frequently trading off object fidelity for size. The original object contains the full information of the content, and usually corresponds to the original version or the most detailed version of the object. The original version of object can be transcoded to a less detailed one, and such a transcoded object is called the transcoded version or the less detailed version of the object. Without loss of generality, we assume that each object can be represented in n versions. The original version of object is denoted as V_1 (i.e., version 1), whereas the least detailed version which cannot be transcoded any more is denoted as V_n (i.e., version n). The intermediate versions are, in turn, denoted as $V_2, ..., V_{n-1}$, in which V_i is a more detailed version than V_j for each i, j if $i < j$.

It is noted that not every V_i can be transcoded to V_j for $i < j$. It is possible that V_i does not contain enough content information for the transcoding to V_j. The transcoding proxy thus must a priori know the transcodable relationship of an object whenever it performs the transcoding. To this end, we devise the notion of *weighted transcoding graph* as defined in Definition 1.

Definition 1 *The weighted transcoding graph, G_i, is a directed graph with weight funtion w_i. G_i depicts the transcoding relationship among transcodable versions of object i. For each vertex $v \in V[G_i]$, v represents a transcodable version of object i. Version u of object i is transcodable to version v iff there exists a directed edge $(u, v) \in E[G_i]$. The transcoding cost from version u to v is given by $w_i(u, v)$ which is the weight of the edge from u to v.*

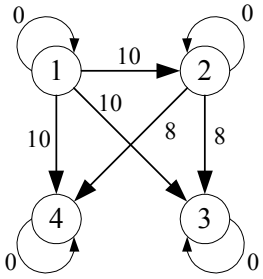

Figure 3. Weighted transcoding graph.

The weighted transcoding graph is in essence an extension to the *transcoding relation graph* proposed in [5]. For each object, we maintain a corresponding weighted transcoding graph in the transcoding proxy. Note that since each version is surely transcodable to itself with null transcoding, there is a directed edge pointed from each version to itself with the corresponding weight being 0. An example of *weighted transcoding graph* is given in Figure 3 where the original version V_1 can be transcoded to each

less detailed versions V_2, V_3 and V_4. However, V_3 cannot be transcoded to V_4 due to insufficient content information.

3. Cache Replacement Algorithms for Transcoding Proxies

In Section 3.1, we define the *generalized profit function* which will be used to determine the eviction priorities of the cached objects in our algorithm. To evaluate the aggregate effect properly, we devise *Procedure MATC (standing for Minimal Aggregate Transcoding Cost)* to find the subgraph of the weighted transcoding graph. This subgraph shows the transcoding relationship which minimizes the aggregate transcoding cost when caching a certain set of versions of the object. Utilizing this *generalized profit function* as the eviction function, we design an innovative cache replacement algorithm for transcoding proxies in Section 3.2. This algorithm is referred to as algorithm *AE (standing for Aggregate Effect)* for the reason that it explores the aggregate effect of caching multiple versions of an object in the transcoding proxy.

3.1. Generalized Profit Function

As mentioned in Section 1, most cache replacement algorithms employ an *eviction function* to determine the priorities of objects to be replaced. In our algorithm, we determine the eviction priorities according to the *generalized profit function*, which will be defined later. Initially, we will derive the individual profit function for the calculation of the profit from caching a single version of an object. Then, we derive the aggregate profit function for the calculation of the profit from caching multiple versions of an object at the same time. Based on the aggregate profit function, we formulate the marginal profit function for the calculation of the profit from caching a version of an object, given other versions of the object are already cached. Finally, we conclude these profit functions as the *generalized profit function*.

Let $o_{i,j}$ denote version j of object i. The reference rates to different versions of objects are assumed to be statistically independent and denoted by $r_{i,j}$, where $r_{i,j}$ is the mean reference rate to version j of object i. The mean delay to fetch object i from the original server to the transcoding proxy is denoted by d_i, where we define the delay to fetch an object as the time interval between sending the request and receiving the last byte of response. The size of version j of object i is denoted by $s_{i,j}$. The corresponding *weighted transcoding graph* of object i is denoted by G_i, and the transcoding delay of object i from version x to version y is given by the weight on the edge (x, y) in $E[G_i]$ which is denoted by $w_i(x, y)$. A list of the symbols used is give in Table 1. With these given parameters, we can determine the profit from caching version j of object i.

Symbol	*Description*
$o_{i,j}$	version j of object i
$r_{i,j}$	the mean reference rate to version j of object i
d_i	the delay of fetching object i from the original server to the proxy
$s_{i,j}$	the size of version j of object i
G_i	the corresponding transcoding relation graph of object i
$w_i(u, v)$	the weight on each edge (u, v) in G_i

Table 1. A list of the symbols used.

First, we consider the profit from caching a single version of an object in the transcoding proxy (i.e., no other versions are cached). From the standpoint of client users, an optimal cache replacement algorithm is expected to minimize the response time perceived. In other words, an optimal cache replacement algorithm will be designed to maximize the delay saving by caching a certain set of Web objects. Thus, the individual profit from caching version j of object i is defined in terms of the delay saving obtained as below.

Definition 2 *$PF(o_{i,j})$ is a function for the individual profit from caching $o_{i,j}$ while no other version of object i is cached.*

$$PF(o_{i,j}) = \sum_{(j,x)\in E[G_i]} r_{i,x} * (d_i + w_i(1, x) - w_i(j, x)) \quad (1)$$

Note in Eq. (1), each $(j, x) \in E[G_i]$ represents each transcodable version from version j to version x. Expression $d_i + w_i(1, x)$ represents the delay if $o_{i,j}$ is not cached (i.e., the delay of fetching object i from the original server plus the delay of transcoding from the original version to version x). On the other hand, $w_i(j, x)$ represents the delay if $o_{i,j}$ is cached. Thus, $d_i + w_i(1, x) - w_i(j, x)$ is the delay saving from caching $o_{i,j}$ in terms of the subsequent references to $o_{i,x}$. Consider the scenario in Figure 4 for example. Suppose we would like to calculate the individual profit from caching version 2 of object i. The transcodable versions from version 2 is pointed by bold arrows in Figure 4. Without caching version 2, the transcoding proxy has to fetch object i from the original server and to perform transcoding to satisfy the subsequent references to versions 2, 3 or 4. The delays for them in this circumstance are all equal to 30. On the other hand, with caching version 2,

the transcoding proxy only has to perform transcoding from version 2 to versions 2, 3 or 4 to satisfy the subsequent references to them with the corresponding delays being 0, 8 and 8 respectively. Thus, the delay saving from caching version 2 is $30 = 30-0$ for each reference to version 2, $22 = 30-8$ for each reference to version 3, and also $22 = 30 - 8$ for each reference to version 4. Multiplying the delay savings by the reference rates, we can get the individual profit from caching version 2. In this example, suppose the reference rates to versions 1, 2, 3 and 4 are all equal to 10. The delay saving from caching version 2 of object i is thus equal to 740 (i.e., $10 * 30 + 10 * 22 + 10 * 22 = 740$).

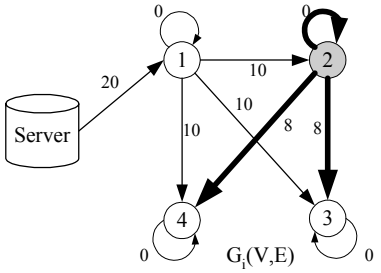

Figure 4. An example illustrating Definition 1.

As a matter of fact, however, the transcoding proxy may cache two or more versions of an object at the same time if the aggregate profit from caching them together is valuable. However, as pointed out in Section 1, the aggregate profit depends on the transcoding relationship among different versions of an object. Hence, we shall devise a procedure, *Procedure MATC (standing for Minimal Aggregate Transcoding Cost)*, to find a subgraph $G_i' \subseteq G_i$ that shows the transcoding relationship which maximizes the aggregate profit when caching two or more versions of an object at the same time.

Procedure MATC(G, w, C)
1 $A \longleftarrow \emptyset$
2 **for** each vertex $u \in C$
3 **do** $color[u] \longleftarrow GRAY$
4 **for** each vertex $v \in V[G]$
5 **do for** each $(x, v) \in E[G]$ **and** $color[x] = GRAY$
6 **do** find the minimum $w(x, v)$
7 $A \longleftarrow A \cup (x, v)$
8 **return** $G'(V[G], A)$

The input of *Procedure MATC* consists of 3 arguments in which G is the corresponding weighted transcoding graph, w is the weights on the edges in G, and C is the set of the versions that the transcoding proxy tries to cache. The operations of *Procedure MATC* can be best understood by the example of Figure 5. Suppose G and w are given by the graph in Figure 3, and the input of C is $\{1, 2\}$. As shown in Figure 5(a), lines 1-3 initialize the set A to the empty set and color vertex 1 and 2 gray, where A is the resulting edge set of the subgraph to be returned. The **for** loop in lines 4-7 finds, for each vertex $v \in V[G]$, the edge (x, v) with minimum weight $w(x, v)$, and adds (x, v) into A. In our example, since (1,1) and (2,2) are the only edges pointed from gray vertexes to themselves, the iterations for vertex 1 and vertex 2 add (1,1) and (2,2) in A as shown in Figure 5(b) and 5(c) respectively. As shown in Figure 5(d), the iteration for vertex 3 adds (2,3) in A because $w(2, 3) < w(1, 3)$. The iteration for vertex 4 works similarly and adds (2,4) in A as shown in Figure 5(e), leading to the resulting subgraph of *Procedure MATC* in Figure 5(f). It can be verified that the time complexity of *Procedure MATC* is $O(VE)$, mainly contributed by the **for** loop in lines 4-7.

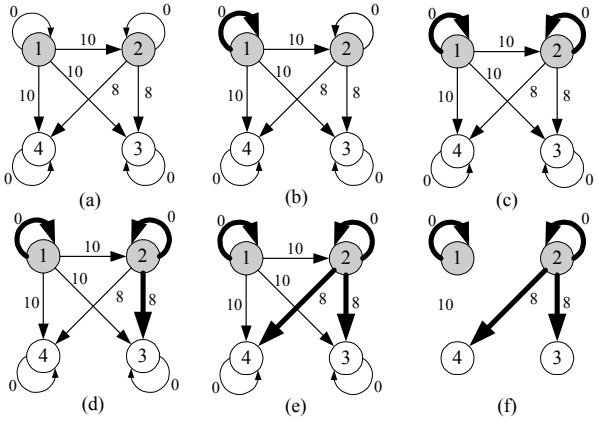

Figure 5. The execution of Procedure MATC.

With *Procedure MATC*, we can thereafter define the aggregate profit from caching multiple versions of an object.

Definition 3 *$PF(o_{i,j1}, o_{i,j2}, ..., o_{i,jk})$ is a function for the calculation of the aggregate profit from caching $o_{i,j1}, o_{i,j2}, ..., o_{i,jk}$ at the same time.*

$$PF(o_{i,j1}, o_{i,j2}, ..., o_{i,jk}) = \sum_{v \in V[G']} \sum_{(v,x) \in E[G']} r_{i,x} * (d_i + w_i(1, x) - w_i(v, x)) \quad (2)$$

where G′ is the resulting subgraph of Procedure MATC(G, w, {j1, j2,..., jk}).

Following the example in Figure 5 and with the resulting subgraph in Figure 5(f), the aggregate profit from caching versions 1 and 2, $PF(o_{i,1}, o_{i,2})$, can be determined as 940 (i.e., $10*(20-0)+10*(20+10-0)+10*(20+10-8)+10*(20+10-8)=940$). After the aggregate profit is calculated, we can determine the marginal profit from caching $o_{i,j}$, given $o_{i,j1}, o_{i,j2}, ..., o_{i,jk}$ are already cached.

Definition 4 *$PF(o_{i,j}|o_{i,j1}, o_{i,j2}, ..., o_{i,jk})$ is a function for the calculation of the marginal profit from caching $o_{i,j}$, given $o_{i,j1}, o_{i,j2}, ..., o_{i,jk}$ are already cached where $j \neq j1, j2, ..., jk$.*

$$PF(o_{i,j}|o_{i,j1}, ..., o_{i,jk}) = PF(o_{i,j}, o_{i,j1}, ..., o_{i,jk}) - PF(o_{i,j1}, o_{i,j2}, ..., o_{i,jk})$$

From the above example, we can obtain that the marginal profit from caching version 1, given version 2 is already cached, is only 200 (i.e., $940 - 740 = 200$). Finally, we can integrate all the profit functions into a *generalized profit function* as

$$PF^G(o_{i,j}) = \begin{cases} \frac{PF(o_{i,j})}{s_{i,j}} \\ \text{if no other version of object } i \text{ cached;} \\ \\ \frac{PF(o_{i,j}|o_{i,j1},...,o_{i,jk})}{s_{i,j}} \\ \text{if there are other versions } o_{i,j1}, ..., o_{i,jk} \text{ cached.} \end{cases} \quad (3)$$

It is noted that the *generalized profit function* is further normalized by the size of version j of object i to reflect the object size factor. The *generalized profit function* defined in Eq. (3) explicitly considers the new emerging factors in the transcoding proxy and the aggregate effect of caching multiple versions of an object. As such, we can evaluate the profit from caching a certain version of an object, and then, in view of the cost model, develop the optimal cache replacement algorithm.

3.2. Design of Algorithm AE

In Section 3.1, we have formulated the *generalized profit function*, PF^G. Based on this *generalized profit function*, we design algorithm *AE* in this subsection. The main idea behind algorithm *AE* is to first order the objects according to their values of the *generalized profit function*, and then select the object with the highest value, one by one, until there is not enough space to hold any object more. The objects selected are thus the objects to be cached in the transcoding proxy, and the rest objects are to be evicted.

Note, however, that some assumptions made in Section 3.1 to facilitate our presentation may have to be relaxed when an algorithm is designed. First, the mean reference rates of Web objects are not static but may change as time advances. This phenomenon is quite common, especially in a news Web site where the reference rates to the news documents decrease as time goes by. Second, the mean reference rates to different Web objects are not independent from one to another. For instance, if there is a hyperlink from one Web object to another, the mean reference rates to them are somewhat dependent. Third, the delays of fetching Web objects from the original servers are not known a priori. To address these issues, we shall employ the concept of *sliding average functions* [14][18] to practically estimate the running parameters, d_i and $r_{i,j}$, in the transcoding proxy in this paper. With the estimated running parameters, we can devise the pseudo-code of algorithm *AE*. Algorithm *AE* takes 4 arguments as the inputs. C is a priority queue which is used to hold the objects cached in the transcoding proxy. The cached objects in C are maintained by a heap data structure in nondecreasing order according to their values of the *generalized profits*. S_{now} is the accumulated size of the objects currently cached in the priority queue. $o_{i,1}$ and $o_{i,j}$ are the original and transcoded versions to be cached.

Algorithm AE(C, S_{now}, $o_{i,1}$, $o_{i,j}$)
1 insert $o_{i,1}$ and $o_{i,j}$ into C
2 **for** each version of object i in C
3 **do** calculate or revise its generalized profit
4 **BuildHeap**(C)
5 **while** $S_{now} > cacheCapacity$
6 **do** exclude the first object $o_{m,n}$ from C
7 $S_{now} \longleftarrow S_{now} - s_{m,n}$
8 **for** each version of object m
9 **do** revise its generalized profits
10 **BuildHeap**(C)

4. Performance Analysis

In this section, we will evaluate the performance of our cache replacement algorithm by conducting an event-driven simulation. The primary goal of performing an event-driven simulation is to assess the effect on the performance of our cache replacement algorithm by varying different system parameters. We will describe the simulation model in Section 4.1. The experimental results will be shown in Section 4.2.

4.1. Simulation Model

The simulation model is designed to reflect the system environment of the transcoding proxy as described in Section 2. In the client model, as in [5], we assume that the

mobile appliances can be classified into five classes. That is, all Web objects could be transcoded to five different versions by the transcoding proxy to satisfy the users' need. Without loss of generality, the sizes of the five versions are assumed to be 100%, 80%, 60%, 40% and 20% of the original object size. Also, we assume a more detailed version can be transcoded to a less detailed one, and the transcoding delay is determined as the quotient of the object size to the transcoding rate. The distribution of these five classes of mobile clients is modeled as a device vector of <15%, 20%, 30%, 20%, 15%>.

Next, we consider the workload of the mobile clients' requests. We assume that the number of total Web objects is 1000. The sizes of the objects are lognormally distributed with the mean value of 15 Kbytes. The popularity of the Web objects follows a Zipf-like distribution, which is widely adopted to be a model for real Web traces [3][4][13] [14]. The default value of parameter α in Zipf-like distribution is set to be 0.75 with a reference to the analyses of real Web traces in [4]. As shown in [8][11], small files are much more frequently referenced than large files. Hence, we assume that there is negative correlation between the object size and its popularity. Finally, the interarrival time of two consecutive clients' requests is modeled by exponential distribution with the mean value of 0.1 second. As to the model of the transcoding proxy, we choose the default cache capacity to be $0.01 * (\sum$object size$)$. The delays of fetching objects from Web servers are given by an exponential distribution. Table 2 provides a summary of the parameters used in the simulation model.

Parameter	*Dist./Default value*
Number of Web objects	1000
Object size	Lognormal ($\mu = 20$)
Object popularity	Zipf-like ($\alpha = 0.75$)
Interarrival time of clients' requests	Exp. ($\mu = 0.1$)
Cache capacity	$0.01 * (\sum$object size$)$
Object fetching delay	Exp. ($\mu = 2$)
Transcoding rate	20 KB/sec

Table 2. Parameters of the simulation model.

4.2. Experimental Results

Based on the simulation model devised, we implement a simulator using Microsoft Visual C++ package on IBM compatible PC with Pentium III 450 CPU and 128 Mbytes RAM. Each set of the experimental results is obtained by ten simulation runs with different seeds. The primary performance metric employed in our experiments is the *delay saving ratio* which is defined as the ratio of total delay saved from cache hits to the total delay incurred under the circumstance without caching. In addition, we also use the *exact hit ratio*, the *useful hit ratio*, and the *overall hit ratio* as the secondary performance metrics in our experiments. The exact hit ratio corresponds to the fraction of requests which are satisfied by the exact versions of the objects cached (i.e., no transcoding is needed for such requests), whereas the useful hit ratio is the fraction of requests which are satisfied by the more detailed versions of the objects cached (i.e., additional transcoding is required for them). Our yardsticks are the *coverage-based* and *demand-based* extensions to the traditional LRU and LRUMIN cache replacement algorithms. We denote the *coverage-based* LRU as LRU_{CV} and the *demand-based* LRU as LRU_{DM}. The *coverage-based* and *demand-based* extensions to LRUMIN are denoted as $\text{LRU}_{\text{CV}}^{\text{MIN}}$ and $\text{LRU}_{\text{DM}}^{\text{MIN}}$ respectively. In addition, to understand the benefit of algorithm *AE* achieved by considering the aggregate effect, we also compare algorithm *AE* with algorithm AE_0 where algorithm AE_0 is a modified version of algorithm *AE*, in which the eviction priorities are determined by the profit function in Definition 1 instead of the *generalized profit function*. Recall that the profit function in Definition 1 did not consider the aggregate effect in the environment of transcoding proxy. By comparing algorithm *AE* with AE_0, the advantage of considering the aggregate effect can be observed. A list of the symbols used is given in Table 3.

Symbol	*Description*
LRU_{CV}	*coverage-based* LRU
LRU_{DM}	*demand-based* LRU
$\text{LRU}_{\text{CV}}^{\text{MIN}}$	*coverage-based* LRUMIN
$\text{LRU}_{\text{DM}}^{\text{MIN}}$	*demand-based* LRUMIN
AE	Algorithm *AE*
AE_0	Modified algorithm *AE* without consideration of the aggregate effect
γ_{FD}	The ratio of the mean object fetching delay to the mean transcoding delay

Table 3. A list of the symbols used.

4.2.1. Impact of Cache Capacity

In the first experiment set, we investigate the performance of our cache replacement algorithm by varying the cache capacity. The simulation results are shown in Figure 6 and Figure 7. As presented in Figure 6, our algorithm outperforms the other ones in terms of the primary performance metric, i.e., the delay saving ratio (DSR). Specifically, the

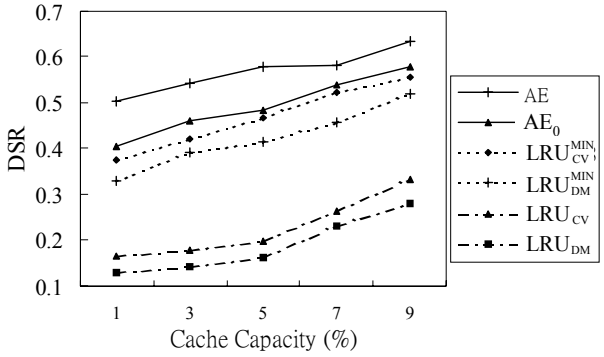

Figure 6. DSR under various cache capacity.

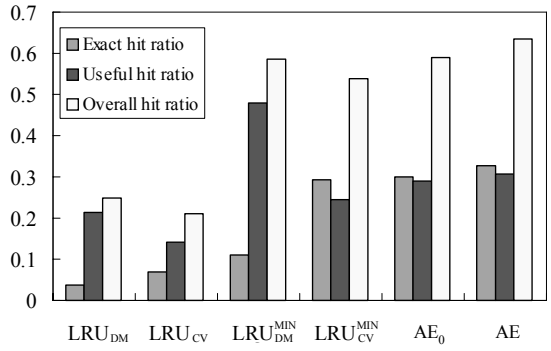

Figure 7. Hit ratios of various algorithms.

mean improvements of the delay saving ratios over LRU_{CV} and LRU_{DM} algorithms are 162.4% and 221.3% respectively, whereas the mean improvements over the LRU_{CV}^{MIN} and LRU_{DM}^{MIN} are 22.4% and 36.4%. The main reason that the extensions to LRU algorithm did not perform well is that the LRU algorithm considers neither the sizes of objects nor the delays of fetching objects from original servers. This drawback is more severe when it comes to the circumstance of the transcoding proxy. On the other hand, the advantage of algorithm *AE* over LRU_{CV}^{MIN} and LRU_{DM}^{MIN} is contributed by the devised *generalized profit function* which explicitly considers the new emerging factors in the environment of transcoding proxies, especially the aggregate effect when observing the difference between AE and AE_0.

Figure 7 provides more insights of the relationship between algorithms and performance. Observe that the exact hit ratios of the *demand-based* algorithms are higher than those of the *coverage-based* algorithms. Conversely, the useful hit ratios of the *coverage-based* algorithms are higher than those of the *demand-based* algorithms. This can be explained by the reason that *demand-based* algorithms always cache the transcoded versions to avoid repeating transcoding operations, thus leading to higher exact hit ratios. However, the *coverage-based* algorithms always cache the original versions to provide maximum coverage on subsequent requests, accounting for the reason that they perform better in terms of useful hit ratios. Although algorithm *AE* is designed to maximize the DSR rather than the hit ratios, algorithm *AE* still outperforms others in terms of the overall hit ratio.

4.2.2. Impact of γ_{FD} Ratio

We define the ratio of the mean object fetching delay to the mean transcoding delay as the γ_{FD} ratio. Formally, the value of γ_{FD} can be determined by

$$\gamma_{FD} = \frac{mean\ fetching\ delay}{mean\ object\ size/transcoding\ rate}.$$

The second experiment set examines the performance of our algorithm by varying the value of γ_{FD}. The simulation results are given in Figure 8 and Figure 9. As shown in Figure 8, the overall hit ratio of each algorithm almost remains the same for various γ_{FD} ratios. This is because all simulation parameters except the mean fetching delay are fixed, and the mean fetching delay will not affect the overall hit ratio. However, the delay saving ratio increases as the γ_{FD} ratio increases in Figure 9. In addition, the performance difference between the *coverage-based* algorithm and the *demand-based* algorithms (e.g. LRU_{CV} and LRU_{DM}) increases at the same time. The former phenomenon is more intuitive because the delay saving ratio is, of course, affected by the mean fetching delay. We explain the latter

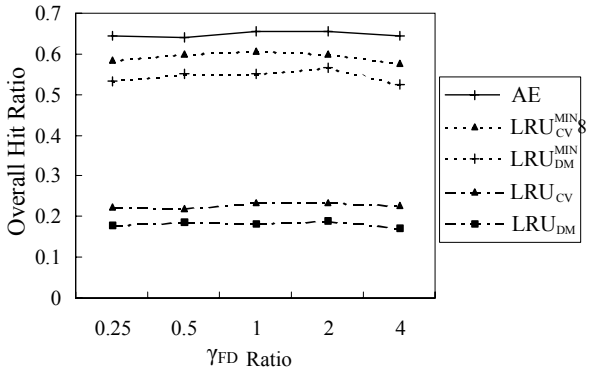

Figure 8. Hit ratios under various γ_{FD} ratios.

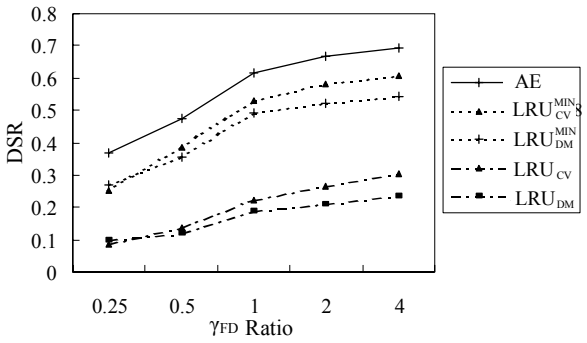

Figure 9. DSR under various γ_{FD} ratios.

phenomenon as follows. The delay saving ratio is contributed by cache hits which can be classified into exact cache hits and useful cache hits. Although the ratios of exact and useful cache hits do not change by γ_{FD} ratio, the weights of their contribution to DSR do change. When the γ_{FD} ratio is small, the delay incurred by the transcoding is more significant than the delay incurred by fetching from the original server. In other words, the contribution of the exact hit ratio is more weighted than that of the useful hit ratio. Thus, the *demand-based* algorithm which avoids repeating transcoding operations performs relatively well. The situation is reversed when the value of the γ_{FD} ratio is large. Unlike either the *demand-based* algorithms which maximize the exact hit ratio or the *coverage-based* ones which maximize the useful hit ratio, algorithm *AE* deals with the caching problem of transcoding proxies in a comprehensive way. Consequently, the performance of algorithm *AE* remains robust against various values of γ_{FD}.

5. Conclusions

In this paper, we developed an efficient cache replacement algorithm for transcoding proxies. We formulated a *generalized profit function* to evaluate the profit from caching each version of object. Based on the weighted transcoding graph and the generalized profit function, we proposed algorithm *AE* as the new cache replacement algorithm for transcoding proxies. Using an event-driven simulation, we showed that algorithm *AE* consistently outperforms companion schemes in terms of the delay saving ratios and cache hit ratios.

Acknowledgment

The authors are supported in part by the National Science Council, Project No. NSC 89-2219-E-002-007 and NSC 89-2213-E-002-032, Taiwan, Republic of China.

References

[1] Akamai Technologies, Inc. http://www.akamai.com.

[2] M. Abrams, C. Standridge, G. Abdulla, S. Williams, and E. Fox. Caching Proxies: Limitations and Potentials. In *Proc. Fourth Int'l World Wide Web Conf.*, Boston, 1995.

[3] C. Aggarwal, J. L. Wolf, and P.-S. Yu. Caching on the World Wide Web. *IEEE Trans. on Knowledge and Data Engineering*, 11(1):94–107, 1999.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web Caching and Zipf-like Distributions: Evidence and Implications. In *Proc. IEEE INFOCOM 1999*, March 1999.

[5] V. Cardellini, P.-S. Yu, and Y.-W. Huang. Collaborative Proxy System for Distributed Web Content Transcoding. In *Proc. ACM CIKM*, pages 520–527, 2000.

[6] S. Chandra, C. Ellis, and A. Vahdat. Application-Level Differentiated Multimedia Web Services Using Quality Aware Transcoding. *IEEE Journal on Selected Areas in Communications*, 2000.

[7] S. Chandra and C. S. Ellis. JPEG Compression Metric as a Quality-Aware Image Transcoding. In *Proc. USENIX 2nd Symp. On Internet Techonlogy and Systems*, pages 81–92, 1999.

[8] C. Cunha, A. Bestavros, and M. Crovella. Characteristics of WWW Client-based Traces. Technical report, Boston univ., April 1995.

[9] R. Floyd and B. Housel. Mobile Web Access Using eNetwork Web Express. *IEEE Personal Communications*, 5(5):47–52, 1998.

[10] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-Based Scalable Network Services. In *Proc. 16th ACM Symp. On Operating Systems Principles*, pages 78–91, Saint-Malo, France, 1997.

[11] S. Glassman. A Caching Relay for the World Wide Web. *Computer Networks and ISDN Systems*, 27, 1994.

[12] R. Mohan, J. R. Smith, and C.-S. Li. Adapting Multimedia Internet Cotent for Universal Access. *IEEE Trans. on Multimedia*, 1(1):104–114, 1999.

[13] V. Padmanabhan and L. Qiu. The Content and Access Dynamics of a Busy Web Site: Findings and Implications. In *Proc. ACM SIGCOMM'00*, 2000.

[14] J. Shim, P. Scheuermann, and R. Vingralek. Proxy Cache Algorithms: Design, Implementation, and Performance. *IEEE Trans. on Knowledge and Data Engineering*, 11(4):549–561, 1999.

[15] WAP Forum. Wireless Application Protocol Cache Model Specification, 1998.

[16] WAP Forum. Wireless Application Group User Agent Profile Specification, 1999.

[17] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. Fox. Removal Policies in Network Caches for World Wide Web Documents. In *Proc. ACM SIGCOMM*, pages 293–304, 1996.

[18] J. Xu, Q. Hu, D.-L. Lee, and W.-C. Lee. SAIU: An Efficient Cache Replacement Policy for Wireless On-demand Broadcasts. In *Proc. ACM CIKM*, pages 46–53, 2000.