An Efficient Cache Replacement Algorithm for Multimedia Object Caching

Keqiu Li[†,‡], Takashi Nanya[‡], Hong Shen[◊], Francis Y. L. Chin[≀], and Weishi Zhang[†]

[†] College of Computer Science and Technology
Dalian Maritime University
No 1, Linghai Road, Dalian, 116026, China

[‡] Research Center for Advanced Science and Technology
The University of Tokyo
4-6-1 Komaba, Meguro-ku, Tokyo, 153-8904, Japan

[◊] Graduate School of Information Science
Japan Advanced Institute of Science and Technology
1-1, Asahidai, Nomi, Ishikawa, 923-1292, Japan

[≀]Department of Computer Science and Information Systems
University of Hong Kong, Pokfulam Road, Hong Kong

Abstract

Multimedia object caching, by which the same multimedia object can be adapted to diverse mobile appliances through the technique of transcoding, is an important technology for improving the scalability of web services, especially in the environment of mobile networks. In this paper, we address the problem of cache replacement for multimedia object caching by exploring the minimal access cost of caching any number of versions of a multimedia object. We first present an optimal solution for calculating the minimal access cost of caching any number of versions of the same multimedia object and its extensive analysis. The performance objective is to minimize the total access cost by considering both transmission cost and transcoding cost. Based on this optimal solution, we propose an efficient cache replacement algorithm for multimedia object caching. Finally, we evaluate the performance of the proposed algorithm with a set of carefully designed simulation experiments for various performance metrics over a wide range of system parameters. The simulation results show that our algorithm outperforms comparison algorithms in terms of all the performance metrics considered.

Keywords: Multimedia object caching, cache replacement, transcoding, mobile network, Internet.

1 Introduction

The World Wide Web has become the most successful application on the Internet since it provides a simple way to access a wide range of information and services. However, due to the dramatic growth in demand, considerable access latency is often experienced in retrieving web objects from the Internet, and popular web sites are suffering from overload. An efficient way to overcome such deficiencies is web caching, by which multiple copies of the same object are stored in geographically dispersed caches.

As many mobile appliances are divergent in size, weight, I/O capabilities, network connectivity, and computing power, differentiated devices should be employed to satisfy their diverse requirements. In addition, different presentation preferences from users make this problem more serious. Transcoding, used to transform a multimedia object from one form to another, frequently through trading off object fidelity for size, is a such technology that can meet these needs [11, 16, 20, 22].

Multimedia object caching, by which the same multimedia object can be adapted to diverse mobile appliances through the technique of transcoding, is an important technology for improving the scalability of web services, especially in the environment of mobile networks. In this paper, we address the problem of cache replacement for multimedia object caching. There are many cache replacement algorithms available in the literature. However, these algorithms cannot be directly applied to solve the problem of cache replacement for multimedia object caching since they consider the case in which the objects are independent while each multimedia object has several versions and all these versions are dependent in that one version may be transformed to another by the technique of transcoding. In [7], the authors proposed a cache replacement algorithm for transcoding proxies (AE for short), which removes objects from the cache according to their generalized profits. When one object is removed from the cache, the generalized profits for the relevant objects will be revised. If the free space cannot accommodate the new object, another object with the least generalized profit is removed until enough room is made for the new object. In this paper, the objects are removed from the cache at the same time according to their generalized composite profits. In Section 3, we will present detailed analysis for computing the composite profit of caching multiple versions of the same multimedia object, which usually differs from simple summation of their individual profits. The following example shows the importance and significance of applying the composite profit for determining the replacement candidates for multimedia object caching.

Example Suppose there are two objects and each object has three versions, i.e., the object set is $o_{1,1}, o_{1,2}, o_{1,3}, o_{2,1}, o_{2,2}, o_{2,3}$, where $o_{i,j}$ denotes version j of object i. We also assume that the size of each object and the generalized profits of caching one or two versions of each object are shown in Table 1, where $s(\cdot)$ and $p(\cdot)$ denote the size and the profit, respectively. For example, the size of $o_{1,2}$ is $s(o_{1,2})$, the generalized profit of caching $o_{2,3}$ is $p(o_{2,3})$, and the generalized profit of caching $o_{1,1}$ and $o_{1,3}$ is $p(o_{1,1}, o_{1,3})$. The profit of caching two versions from

two different objects are the summation of the profit of caching each version since these two versions are independent.

Table 1: Data Used in the Example

$s(o_{1,1})$	$s(o_{1,2})$	$s(o_{1,3})$	$s(o_{2,1})$	$s(o_{2,2})$	$s(o_{2,3})$
3	2	1	3	2	1
$p(o_{1,1})$	$p(o_{1,2})$	$p(o_{1,3})$	$p(o_{2,1})$	$p(o_{2,2})$	$p(o_{2,3})$
18	20	16	16	18	18
$p(o_{1,1}, o_{1,2})$	$p(o_{1,1}, o_{1,3})$	$p(o_{1,2}, o_{1,3})$	$p(o_{2,1}, o_{2,2})$	$p(o_{2,1}, o_{2,3})$	$p(o_{2,2}, o_{2,3})$
29	25	28	26	30	28

If an object with size 2 is to be inserted, it is obvious that object $o_{2,1}$ should be removed because it has the least generalized profit, and its size is enough to accommodate the new object. In this case, AE is efficient. If an object with size 4 is to be inserted, AE will first remove object $o_{2,1}$ from the cache, and then remove $o_{1,3}$ from the cache because these two objects are the ones with the least profits, and the total size of them is enough for the new object. The lost profit by removing these two objects is 32. We can see AE is not efficient in this case because the lost profit is 25 when $o_{1,1}$ and $o_{1,3}$ are removed. The main reason is that the aggregate profit of caching multiple versions of the same object is not the simple summation of that of each version. Furthermore, considering all the objects cached makes this problem more complex. □

From this example, we can see that the composite profit of caching objects plays an important role on deciding the replacement candidates for accommodating a new object due to the limited cache size. If all the objects are dependent, the composite profit is the simple summation of the individual profit. This is not true when multimedia objects are included due to the relationship among different versions of the same multimedia object. Therefore, it is of great importance to involve the composite profit of caching multiple versions of the same multimedia object in designing a cache replacement algorithm for multimedia object caching. From this example, we can also see that the worst case of our method is the same as the method proposed in [7]. The main contributions of this paper are summarized as follows.

- We present an optimal solution for calculating the minimal access cost of caching any number of versions of the same multimedia object and its extensive analysis.
- We propose an efficient cache replacement algorithm for multimedia object caching by utilizing the above optimal solution.
- We evaluate our algorithm on various performance metrics through extensive simulation experiments. The implementation results show that our algorithm outperforms existing algorithms.

The rest of this paper is organized as follows. Section 2 introduces related work. In Section 3, we present an efficient cache replacement algorithm for multimedia object replacement. The simulation model and performance evaluation are described in Sections 4 and 5, respectively. Section 6 summarizes our work and concludes the paper.

2 Related Work

Cache replacement plays a significant role on the functionality of web caching. A number of cache replacement algorithms have been proposed in the literature with the purpose of attempting to minimize various cost metrics, such as hit rate, byte hit rate, average access latency, and total access cost. All these algorithms can be generally classified into such categories as deterministic policies [8, 21], greedydual-based policies [2, 10], hybrid policies [9, 14], randomized policies [13, 18]. An overview of web caching algorithms can be found in [3]. However, all these algorithms consider the case in which web objects are independent. The objects addressed in this paper are multimedia objects; thus, several different versions of the same multimedia object are dependent through the technology of transcoding.

There is little work done on finding efficient cache replacement algorithms for multimedia object caching. In [16, 19], the authors studied several caching strategies or architectures for transcoding proxies. However, all these strategies or architectures are evolved from the algorithms mentioned above and the authors have not considered the aggregate effect of caching multiple versions of the same multimedia object at the same time. The algorithm proposed by Chang et al. in [7] is on the similar line with our algorithm. However, Chang's algorithm is not efficient, which has been shown by the example given in the previous section.

3 An Efficient Cache Replacement algorithm for Multimedia Object Caching

In this section, we first introduce some notations and definitions, and then present an optimal solution for calculating the minimal access cost of caching any number of a multimedia object. Finally, we propose an efficient cache replacement algorithm for multimedia object caching.

In this paper, we use o_i to denote a multimedia object i, m_i to denote the number of versions owned by o_i, $o_{i,j}$ to denote version j of o_i, and $s_{i,j}$ to denote the size of $o_{i,j}$. The relationship among different versions of a multimedia object can be expressed by a weighted transcoding graph [7]. An example of such a graph is shown in Figure 1, where object o_1 has five versions $o_{1,1}, o_{1,2}, o_{1,3}, o_{1,4}, o_{1,5}$.

We can see that the original version $o_{1,1}$ can be transcoded to each of the less detailed versions $o_{1,2}$, $o_{1,3}$, $o_{1,4}$, and $o_{1,5}$. It should be noted that not every version can be transcoded to another version since it is possible that not enough content information is contained for the

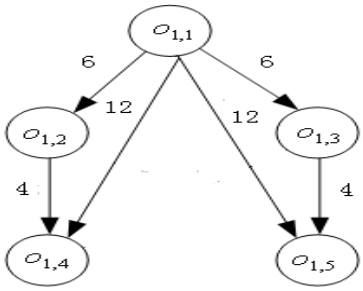

Figure 1: A Weighted Transcoding Graph

transcoding from one version to another. The transcoding cost for a multimedia object from o_{i,j_1} to o_{i,j_2} is denoted by $t(o_{i,j_1}, o_{i,j_2})$. Obviously, $t(o_{i,j}, o_{i,j}) = 0$. If a version cannot be transcoded from another version, we consider the transcoding cost as infinity. For example, in Figure 1, $t(o_{1,1}, o_{1,2}) = 6$, $t(o_{1,1}, o_{1,5}) = 12$, $t(o_{1,2}, o_{1,4}) = 4$, and $t(o_{1,4}, o_{1,5}) = \infty$. For cache replacement, only the node with a cache (denoted by v) and the server (or a cache) that holds a version of a multimedia object (denoted by u) are considered. $L_{i,j}$ is used to denote the cost of sending a request for $o_{i,j}$ and the relevant response over the link (u, v) and called transmission cost in this paper. In this paper, we assume that $L_{i,j} = L_i$, i.e., the transmission cost is the same for each version on the link (u, v). Let $f_{i,j}$ be the access frequency for $o_{i,j}$ from node v. In this paper, we assume that the transcoding graph is a linear array and the transcoding cost between any two adjacent versions is constant, i.e., $t(o_{i,j_1}, o_{i,j_2}) = \sum_{n=j_1}^{j_2-1} t(o_{i,n}, o_{i,n+1}) = (j_2 - j_1)^+ T_i$, where $x^+ = x$ if $x \geq 0$ else $x^+ = \infty$. Our analysis can be easily extended to the general transcoding graph. We can see that there should exist some positive integer δ such that $(\delta - 1)T_i \leq L_i$, and $\delta T_i > L_i$. If there does not exist such a δ, i.e., $L_i \gg T_i$ or $T_i \gg L_i$, obviously, these are two trivial cases. If $L_i \gg T_i$, then we should leave the most detailed versions of different multimedia objects in the cache so that no transmission cost is necessary to occur. If $T_i \gg L_i$, then we can apply the general cache replacement algorithm (e.g. *LRU*) to solve the problem of cache replacement for multimedia object caching since all the versions of the same multimedia object can be viewed as independent objects in this case.

Now we begin to discuss the access cost of caching k versions of multimedia object o_i (AC-k problem for short), where $1 \leq k \leq m_i$ and m_i is the number of versions owned by o_i.

First, we begin by computing the access cost of caching only one version $o_{i,j}$ at node v with $1 \leq j \leq m_i$. Intuitively, all the requests for version $o_{i,j'}$ with $j' < j$ will be handled by server u, while some of the requests for $o_{i,j'}$ with $j' \geq j$, depending on the transcoding cost and the transmission cost, will be taken care of by transcoding from version $o_{i,j}$. Therefore, the total

access cost of caching only version $o_{i,j}$ at node v is computed as follows:

$$C_{1,m_i}(o_{i,j}) = \sum_{n=1}^{j-1} f_{i,n} L_i + \sum_{n=j}^{m_i} f_{i,n} \min\{(n-j)T_i, L_i\} \tag{1}$$

where $C_{1,m_i}(o_{i,j})$ is the total access cost of caching only version $o_{i,j}$ at node v with $1 \leq j \leq m_i$.

Since version $o_{i,j}$ is cached at node v, we can see that δ is such a parameter that the request for version $o_{i,n}$ will be served by the local node if $0 < n - j < \delta$, and the request for version $o_{i,n}$ will be served by the server if $n - j \geq \delta$. Thus, based on Equation (1), $C_{1,m_i}(o_{i,j})$ can be further defined as follows:

$$C_{1,m_i}(o_{i,j}) = \begin{cases} \sum_{n=1}^{j-1} f_{i,n} L_i + \sum_{n=j}^{j+\delta-1} f_{i,n}(n-j)T_i + \sum_{n=j+\delta}^{m_i} f_{i,n} L_i \ (j+\delta \leq m_i) \\ \sum_{n=1}^{j-1} f_{i,n} L_i + \sum_{n=j}^{m_i} f_{i,n}(n-j)T_i \ (j+\delta > m_i) \end{cases} \tag{2}$$

It is easy to see that $C_{1,m_i}(o_{i,1})$ can be calculated in $O(m_i)$ time. Thus, $C_{1,m_i}(o_{i,2})$, $C_{1,m_i}(o_{i,3})$, $\cdots$, $C_{1,m_i}(o_{i,m_i})$ can all be done in constant time. Therefore, based on the cost function as given in Equation (1), the AC-1 problem of caching only one version $o_{i,j}$ can be solved in $O(m_i)$ time.

The second step is to extend the above solution to compute the optimal solution for caching two versions, o_{i,j_1} and o_{i,j_2}, at the same time at node v.

Suppose that o_{i,j_1} and o_{i,j_2} are the two optimal versions to be cached. The key observation here is that o_{i,j_1} is also an optimal solution for the problem with $1 \leq j_1 \leq j_2$ if $j_1 < j_2$, because the requests for $\{o_{i,j_2}, o_{i,j_2+1}, \cdots, o_{i,m_i}\}$ can only be served by o_{i,j_2}. Regarding to this observation, we have the following lemma.

Lemma 1 *Assume that o_{i,b_p} and o_{i,b_q} are the optimal solutions for the problem of caching only one version from the set of $\{o_{i,1}, o_{i,2}, \cdots, o_{i,p-1}\}$ and $\{o_{i,1}, o_{i,2}, \cdots, o_{i,q-1}\}$ respectively, then we have $b_p \leq b_q$ if $p < q$.*

Proof Without loss of generality, it is sufficient for us to prove that $b_p \leq b_{p+1}$ where $1 \leq b_p \leq p-1$ and $1 \leq b_{p+1} \leq p$. The proof is by contradiction. Assume that we have $b_p > b_{p+1}$. As o_{i,b_p} is the optimal version to be cached, we have $C_{1,p}(o_{i,b_p}) < C_{1,p}(o_{i,b_{p+1}})$. From the definition of the access cost function $C_{1,p}$ as given in Equation (1), adding $o_{i,p}$ to the set $\{o_{i,1}, o_{i,2}, \cdots, o_{i,p-1}\}$ will increase both $C_{1,p}(o_{i,b_p})$ and $C_{1,p}(o_{i,b_{p+1}})$ by $f_{i,p} \min\{(p-b_p)T_i, L_i\}$ and $f_{i,p} \min\{(p-b_{p+1})T_i, L_i\}$ respectively. The increase to $C_{1,p}(o_{i,b_{p+1}})$ is no less than that to $C_{1,p}(o_{i,b_p})$ because $b_p > b_{p+1}$. So we have $C_{1,p+1}(o_{i,b_p}) < C_{1,p+1}(o_{i,b_{p+1}})$, which contradicts the fact that $C_{1,p+1}(o_{i,b_{p+1}})$ is the minimum access cost of caching $o_{i,b_{p+1}}$ for the problem with $\{o_{i,1}, o_{i,2}, \cdots, o_{i,p-1}, o_{i,p}\}$. Hence the lemma is proven. □

Based on Lemma 1, we can see that the feasible range of the optimal solution for the problem with $\{o_{i,1}, o_{i,2}, \cdots, o_{i,q}\}$ can be reduced if the optimal version for the problem with $\{o_{i,1}, o_{i,2}, \cdots, o_{i,p}\}$ has been obtained. So is the other case when the optimal solution for the problem with $\{o_{i,1}, o_{i,2}, \cdots, o_{i,q}\}$ is known, the feasible range of the optimal solution for the problem with $\{o_{i,1}, o_{i,2}, \cdots, o_{i,p}\}$ is also reduced. Therefore, we can find o_{i,b_p} and compute $C_{1,p}(o_{i,p})$ by divide and conquer.

Let $D^{(k)}_{p,q}$ denote the minimum access cost of caching k versions of object o_i for the AC-k problem with $q-p$ versions, i.e., $o_{i,p}, o_{i,p+1}, \cdots, o_{i,q-1}$, where $1 \leq p < q \leq m_i$. Thus, $D^{(1)}_{1,p} = C_{1,p}(o_{i,b_p})$ and $D^{(1)}_{1,m_i+1} = \min\limits_{1 \leq k \leq m_i}\{C_{1,m_i+1}(o_{i,k})\}$. Based on Lemma 1, we have the following theorem on the time complexity of computing $D^{(1)}_{1,p}$ for $1 < p \leq m_i$.

Theorem 1 *All the AC-1 problems for* $\{o_{i,1}, o_{i,2}, \cdots, o_{i,p}\}$ *where* $1 \leq p \leq m_i$, *i.e.,* $D^{(1)}_{1,p}$ *for* $1 < p \leq m_i$, *can be computed in* $O(m_i \log m_i)$ *time.*

Proof Assume that there exists an integer θ such that $m_i = 2^\theta$, then we can compute $D^{(1)}_{1,\frac{1}{2}m_i}$ in $O(m_i)$ time. Assume that $o_{i,b_{\frac{m_i}{2}}}$ is the optimal solution for the problem of caching only one version with $\{o_{i,1}, o_{i,2}, \cdots, o_{i,\frac{m_i}{2}-1}\}$, then we can find the optimal solution for the problem of caching only one version for $\{o_{i,1}, o_{i,2}, \cdots, o_{i,\frac{m_i}{4}}\}$ in $O(m_i)$ time. Similarly, $D^{(1)}_{1,\frac{3m_i}{4}}$ can also be computed by solving the problem of caching only one version with $\{o_{i,1}, o_{i,2}, \cdots, o_{i,\frac{3m_i}{4}-1}\}$. As we have already computed $C_{1,\frac{m_i}{2}}(o_{i,y})$ where $y = \min(b_{\frac{m_i}{2}}, \frac{m_i}{2}-1)$, we can base on this result to compute $C_{1,\frac{3m_i}{4}}(o_{i,y})$ for $\{o_{i,1}, o_{i,2}, \cdots, o_{i,\frac{3m_i}{4}-1}\}$ (by adding at most $\frac{m_i}{4}$ terms to $C_{1,\frac{m_i}{2}}(o_{i,\frac{m_i}{2}-1})$. We then compute $C_{1,\frac{3m_i}{4}}(o_{i,y}), C_{1,\frac{3m_i}{4}}(o_{i,y+1}), \cdots, C_{1,\frac{3m_i}{4}}(o_{i,\frac{3m_i}{4}-1})$ in at most $O(\frac{3m_i}{4}-y)$ time. So it takes at most $O(m_i)$ time to compute $D^{(1)}_{1,\frac{m_i}{4}}$ and $D^{(1)}_{1,\frac{3m_i}{4}}$. According to the similar decomposition, $D^{(1)}_{1,\frac{m_i}{8}}$, $D^{(1)}_{1,\frac{3m_i}{8}}$, $D^{(1)}_{1,\frac{5m_i}{8}}$, and $D^{(1)}_{1,\frac{7m_i}{8}}$ can all be solved in $O(m_i)$ time. After repeating this process $\log m_i$ times, we can finish computing $D^{(1)}_{1,p}$ for $1 < p \leq m_i$. Hence, the theorem is proven. □

Now we can accomplish the problem of caching two versions in the following three steps.

- *Step* 1: Evaluate $D^{(1)}_{1,p}$ for $1 < p \leq m_i$, where $D^{(1)}_{1,p}$ denotes the minimum access cost of caching only one version for the AC-1 problem with $p-1$ versions, i.e., $o_{i,1}, o_{i,2}, \cdots, o_{i,p-1}$. In particular, $D^{(1)}_{1,m_i+1} = \min\limits_{1 \leq k \leq m_i}\{C_{1,m_i+1}(o_{i,k})\}$.

- *Step* 2: Evaluate $\overline{D}_p$ for $2 \leq p \leq m_i$, where $\overline{D}_p$ is the access cost for versions $o_{i,p}, o_{i,p+1}, \cdots, o_{i,m_i}$ if $o_{i,p}$ is cached at node v. $\overline{D}_p$ is defined as follows:

$$\overline{D}_p = \begin{cases} \sum\limits_{j=p}^{p+\delta-1} f_{i,j}(j-p)T_i + \sum\limits_{j=p+\delta}^{m_i} f_{i,j}L_i & \text{if } p+\delta \leq m_i \\ \sum\limits_{j=p}^{m_i} f_{i,j}(j-p)T_i & \text{if } p+\delta > m_i \end{cases}$$

- *Step* 3: Compute $D^{(2)}_{1,m_i}$, where $D^{(2)}_{1,m_i}$ is the minimum access cost of caching two versions for the problem with $\{o_{i,1}, o_{i,2}, \cdots, o_{i,m_i}\}$. $D^{(2)}_{1,m_i}$ is calculated as follows:

$$D^{(2)}_{1,m_i} = \min_{2 \le p \le m_i} \{D^{(1)}_{1,p} + \overline{D}_p)\}$$

It is easy to show that $D^{(2)}_{1,m_i}$ is the minimum access cost of caching two versions for the AC-2 problem and the time complexity of computing $D^{(2)}_{1,m_i}$ is $O(m_i \log m_i)$.

After we have calculated $D^{(1)}_{1,p}$ for $1 \le p \le m_i$ in *Step* 1, we can obtain $D^{(2)}_{1,p}$ for all $2 \le p \le m_i$ in another $O(m_i \log m_i)$ time by divide and conquer, where $D^{(2)}_{1,p}$ is the minimum access cost of caching only two versions for the problem with $p-1$ versions, i.e., $o_{i,1}, o_{i,2}, \cdots, o_{i,p-1}$. The main idea is similar to Lemma 1 in the finding of $D^{(1)}_{1,p}$. Assume that $o_{i,b_{p_1}}$ and $o_{i,b_{p_2}}$ with $1 \le b_{p_1} < b_{p_2} < p$ are the two optimal versions cached in node v for $o_{i,1}, o_{i,2}, \cdots, o_{i,p-1}$ to achieve the optimal access cost $D^{(2)}_{1,p}$. Similarly, $o_{i,b_{q_1}}$ and $o_{i,b_{q_2}}$ with $1 \le b_{q_1} < b_{q_2} < q$ are the two optimal versions cached in node v for $o_{i,1}, o_{i,2}, \cdots, o_{i,q-1}$ to achieve the optimal access cost $D^{(2)}_{1,q}$. We can show with a similar argument with Lemma 1 that $b_{p_2} \le b_{q_2}$ if $p < q$ and this property limits the range of searching for the optimal solutions. As in Theorem 1, the two optimal solutions in $D^{(2)}_{1,\frac{m_i}{2}}$ can be found in $O(m_i)$ time after knowing the optimal versions of $D^{(1)}_{1,p}$ for $1 < p \le m_i$; then $D^{(2)}_{1,\frac{m_i}{4}}$ and $D^{(2)}_{1,\frac{3m_i}{4}}$ in another $O(m_i)$ time; then $D^{(2)}_{2,\frac{m_i}{8}}$, $D^{(2)}_{1,\frac{3m_i}{8}}$,$D^{(2)}_{1,\frac{5m_i}{8}}$, and $D^{(2)}_{1,\frac{7m_i}{8}}$ in another $O(m_i)$ time until $D^{(2)}_{1,p}$ for $2 < p \le m_i$ are found after $\log m_i$ times. Therefore, the minimum access cost of caching three versions, denoted by $D^{(3)}_{1,m_i}$, can be computed similarly, i.e., $D^{(3)}_{1,m_i} = \min_{3 \le p \le m_i} \{D^{(2)}_{1,p} + \overline{D}_p)\}$, with at most $O(m_i \log m_i)$ time. Using the same idea, we can solve the problem of caching k versions in $O(km_i \log m_i)$ time with $1 \le k \le m_i$.

Let $D^{(k)}_{1,m_i}$ denote the minimum access cost of caching k versions from m_i versions, i.e., $o_{i,1}, o_{i,2}, \cdots, o_{i,m_i}$, then it is easy to show that $D^{(k)}_{1,m_i}$ can be computed in $O(km_i \log m_i)$ time.

In the following, we propose an efficient cache replacement algorithm for multimedia object caching based on the above optimal solution. Suppose that there are l different multimedia objects cached and the size of a new object to be cached is s, then we should find a subset of objects $O^* \subseteq O$ that satisfies the following conditions.

(1) $\sum_{o_{i,j} \in O^*} s_{i,j} \ge s.$

(2) $(\forall O' \subseteq O$ that satisfies (1)) $CS^G(O^*) \le CS^G(O')$.

where $O^* = \{o_{1,\alpha^1_1}, \cdots, o_{1,\alpha^{r_1}_1}, \cdots, o_{l,\alpha^1_l}, \cdots, o_{l,\alpha^{r_l}_l}\}$ is the set of objects to be removed, $O = \{o_{1,\beta^1_1}, \cdots, o_{1,\beta^{c_1}_1}, \cdots, o_{l,\beta^1_l}, \cdots, o_{l,\beta^{c_l}_l}\}$ is the set of objects cached, and $CS^G(O^*)$ is defined as

the generalized access cost loss [1] and calculated as $CS^G(O^*) = \sum_{i=1}^{l} C_{1,r^i}(o_{i,\alpha_i^1}, \cdots, o_{i,\alpha_i^{r_i}}) / \sum_{o_{i,j} \in O^*} s_{i,j}$. $CS^G(O^{'})$ can be similarly defined. Obviously, (1) is to make enough room for the new object, and (2) is to evict those objects whose generalized access cost loss is minimal.

The naive approach to find such O^* will be in NP hard, same as the packing problem. In the following, we present an algorithm that computes an approximate answer of the problem efficiently by decomposing the set of the candidate objects to be removed into smaller sets and each such set can be decided in polynomial time.

Before we present the algorithm, we introduce some notations. In the following, let $R^*(i,k)$ denote the minimal generalized access cost of caching k versions of object i and $R^*(k)$ the minimal generalized access cost loss of the k objects to be removed. We can see that the k objects to be removed can be k versions of a multimedia object or different versions of different multimedia objects. Thus, k can be decomposed as $k = k_1 + k_2 + \cdots + k_a$, where a is the number of different objects to be removed and $0 \leq k_i \leq k$ is the number of versions of an object that are in the set of the k objects to be removed. For example, $1 \rightarrow \{1+0\}$, $2 \rightarrow \{2+0, 1+1\}$, $3 \rightarrow \{3+0, 2+1, 1+1+1\}$, $4 \rightarrow \{4+0, 3+1, 2+2, 2+1+1, 1+1+1+1\}$, $5 \rightarrow \{4+1, 3+1+1, 3+2, 2+1+1+1, 2+2+1, 1+1+1+1+1\}$, $\cdots$. For the instance of $k = 4$, k can be the combination of $1+1+1+1$, $1+1+2$, $2+2$, $1+3$, and $0+4$, where $1+1+1+1$ means that the objects to be removed should be the first four objects with minimal generalized access cost of caching one version, $1+1+2$ means that the objects to be removed should be the three objects, i.e., the first two objects with minimal generalized access cost of caching one version and the last object with minimal generalized access cost of caching two versions, etc. It can be easily proved that there are at most k^2 different such combinations in all. Therefore, we have $R^*(k) = \min\{R^*(1,k), R^*(2,k), \cdots, R^*(l,k), \min_{k=k_1+k_2+\cdots+k_a} \{R^*(k_1) + R^*(k_2) + \cdots + R^*(k_a)\}\}$.

We denote the set of all the objects that achieves $R^*(k)$ by $O^*(k)$ and their total size is $S^*(k)$. Now we give an example to show how to calculate $R^*(k)$. For the case of $k = 3$, we have the combination of $3 + 0$, $1 + 2$, and $1 + 1 + 1$, each of which can be computed using the previous calculation results. For $3 + 0$, we just choose three versions from one object with minimal generalized access cost of caching three versions. For $1 + 2$, we choose the version from an object with minimal generalized access cost of caching one version and two versions from another object with minimal generalized access cost of caching two versions. When we calculated $R^*(1)$ and $R^*(2)$, they may be using a same version. In this case, we select another version with minimal access cost that is not included. We denote the set of versions calculated by $R^*(1)$ and $R^*(2)$ as $O^*(1)$ and $O^*(2)$, respectively. In this case we will recalculate the set of versions with minimal number of elements by another set of versions of the same object with the same number of elements with more generalized access cost loss. For example, if $o_{1,1} \in O^*(2)$

[1] The generalized access cost loss and the generalized access cost are used interchangeably in this paper since they are defined in the same way.

and $o_{1,1} \in O^*(2)$, then we will recalculate $O^*(1)$, i.e., finding $o_{1,j}$ with the minimal generalized access cost loss except $o_{1,1}$ to represent $o_{1,1}$. Although this will be very costly in theory, the fact that the number of objects we hope to remove in practice is very small makes it feasible. We shall further study this issue in our future work. Based on the above calculation, we finally find how the k objects should be selected such that the generalized access cost loss is minimized. In fact, there may exist a replacement decision by removing more than k objects and the generalized access cost loss is less. Thus, the minimization here is conditional, i.e., under the condition that the minimal number of different objects is to be removed.

With the above analysis, we can devise the pseudocode of our algorithm as follows. In the algorithm, C is used to hold the cached objects, S_c is the cache capacity, S_u is the cache capacity used, o is the object to be cached, and its size is s.

Algorithm MOR (C, S_c, S_u, o)
Input: C, S_c, S_u, o
Output: $O^*(k)$
1. INSERT o INTO C
2. $k = 0$
3. $S^*(k) = 0$
4. WHILE $S_c - S_u - S^*(k) < s$ DO
5. $k = k + 1$
6. FOR $i = 1$ TO l DO
7. CALCULATE $R^*(i, k)$
8. CALCULATE $R^*(k)$
9. CHECK $O^*(k)$ (make all the k objects different)

From Algorithm MOR, we can see that it has 9 loops. Loops $1 - 3$ is the initialization, where Loop 1 is to insert the new object to the cache, Loop 2 is to set the counter k, i.e., the number of the replacement candidates, to be 0, and Loop 3 is to set the initial size of the replacement candidates to be 0. Loops $4 - 8$, i.e., the *while* loop, is executed until the total size of the replacement candidates is enough to accommodate the new object. Loops $6 - 7$, the *for* loop, is to calculate the minimal generalized access cost of caching k objects. The last loop, i.e., Loop 9, is to determine the replacement candidates. As we mentioned previously in this section, it is necessary to check $O^*(k)$ to make all the k objects different. The example shown in Section 1 can also be viewed as an illustrative example for this algorithm.

Regarding to the time complexity of this algorithm, we have the following theorem.

Theorem 2 *The time complexity of Algorithm MOR is $O(k^2(l + k^2) \log (l + k^2))$, where l is the total number of different objects cached and k is the number of objects to be removed.*

Proof Suppose k objects are removed to make room for the new object. The running time of Algorithm MOR mainly depends on Steps 4, 6, 8, and 9. The running time of Step 6 is

determined by computing $R^*(i,k)$ for $1 \leq i \leq l$. For object i, calculating $R^*(i,k)$ is to find the minimal generalized access cost of caching k versions of object i. Note that we should compute the aggregate profit of caching k versions of object i, and then order them according to the calculated profit. Thus, the running time for calculating $R^*(i,k)$ is $O(C(m_i;k)\log C(m_i;k))$. Therefore, The running time of Step 6 is $O(\sum_{i=1}^{l} C(m_i;k)\log C(m_i;k))$ since there are l objects cached and $C(m_i;k) = m_i!/(k!(m_i-k)!)$. The running time for Step 8 is $O((l+k)\log(l+k))$ because we should order all $l+k$ items to find the minimal one among them. Thus, the total running time for Algorithm MOR (Step 4) is $O(\sum_{k=1}^{k^2}[(l+k^2)\log(l+k^2) + \sum_{i=1}^{l} C(m_i;k)\log C(m_i;k)]) = O(k^2(l+k^2)\log(l+k^2))$ since in general $m_i \approx 10$ and l is very very large, where k is the number of objects to be removed, l is the number of different objects, and m_i is the number of versions of object i. Since the running time for Step 9 is $O(\log l)$, the total running time for Algorithm MOR is $O(k^2(l+k^2)\log(l+k^2))$. Hence, the theorem is proven. □

From Theorem 2, we know that the time complexity of Algorithm MOR depends on k, i.e., the number of objects to be removed. In practical execution, we always stop the execution of searching the objects to be removed to make room for the new object when k reaches a certain number. This is based on the fact that it is not beneficial to remove many objects to accommodate only one object. So the practical time complexity of Algorithm MOR is $O(l\log l)$, which is the same as that of the algorithm proposed in [7]. However, from the algorithm we know that we have to search the entire cache for the other versions of the object and then recalculate the generalized access cost for them whenever we insert or evict an object into or from the cache. Such operations are, in general, very costly. Here, we save calculated results for later computation, which will save a lot of computations. For example, after we finish computing $R^*(k)$, we save it using an array. When we hope to compute $R^*(k+1)$, we do not need to recalculate $R^*(k')$ for $1 \leq k' \leq k$ again by reading it from the array directly.

4 Simulation Model

In this section, the simulation model used for performance evaluation is described. We have performed extensive simulation experiments to compare our algorithm with existing algorithms. The system configuration is outlined in Section 4.1, and existing algorithms used for the purpose of comparison are introduced in Section 4.2.

4.1 System Configuration

To the best of our knowledge, it is difficult to find true trace data in the open literature to execute such simulations. Therefore, we generated the simulation model from the empirical results presented in [1, 4–7].

The network topology was randomly generated by the Tier program [6]. Experiments for many topologies with various parameters were conducted and the relative performance of our algorithm was found to be insensitive to topology changes. Here, only the experimental results for one topology are presented due to space limitations. The characteristics of this topology and the workload model are shown in Table 2, which are chosen from the open literature and are considered to be reasonable.

Table 2: Parameters Used in Simulation

Parameter	Value
Number of WAN Nodes	200
Number of MAN Nodes	200
Delay of WAN Links	Exponential Distribution ($\theta = 1.5Sec$)
Delay of MAN Links	Exponential Distribution ($\theta = 0.7Sec$)
Number of Servers	100
Number of Web Objects	1000 objects per server
Web Object Size Distribution	Pareto Distribution ($\mu = 6KB$)
Web Object Access Frequency	Zipf-Like Distribution ($\alpha = 0.7$)
Relative Cache Size Per Node	4%
Average Request Rate Per Node	$U(1, 9)$ requests per second
Transcoding Cost	$50KB/Sec$

The WAN (Wide Area Network) is viewed as a backbone network to which no servers or clients are attached. Each MAN (Metropolitan Area Network) node is assumed to connect to a content server. Each MAN and WAN node is associated with a cache. To generate the workload of clients' requests, we assume that the number of total web objects is N and these N objects are divided into two types: *text* and *multimedia*. In the experiments, the object sizes are assumed to follow a Pareto distribution and the average object size is $6KB$. Similar to the assumption in [7], we also assume that the mobile appliances can be classified into five lasses whose distribution is modeled as a device vector of {10%, 20%, 30%, 25%, 15%}. Without of less generality, the sizes of the five versions of each multimedia object are assumed to be 100 percent, 80 percent, 60 percent, 40 percent, and 20 percent of the original object size. The relationships among the five versions are modeled by the transcoding graph as shown in Fig. 2. The transcoding delay is determined as the quotient of the object size to the transcoding rate. In the simulation, we set the transcoding rate to be $50K$ Bytes per second.

In the experiments, the client at each MAN node randomly generates the requests, and the average request rate of each node follows the distribution of $U(1, 9)$, where $U(x, y)$ represents a uniform distribution between x and y. The access frequencies of both the content servers and

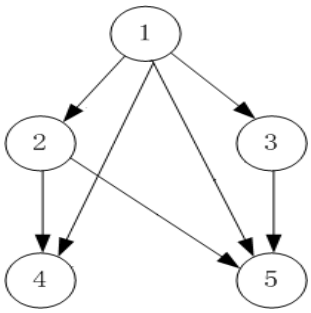

Figure 2: Transcoding Graph for Simulation

the objects maintained by a given server follow a Zipf-like distribution [5, 12]. Specifically, the probability of a request for object O in server S is proportional to $1/(i^{\alpha} \cdot j^{\alpha})$, where S is the ith most popular server and O is the jth popular object in S. The delay of both MAN links and WAN links follows an exponential distribution; the average delay for WAN links is 1.5 seconds and the average delay for MAN links is 0.7 seconds. Similar to the studies in [5, 17], cache size is described as the total relative size of all objects available in the content server.

The cost for each link is calculated by the access delay. For simplicity, the delay caused by sending the request and the relevant response for that request is proportional to the size of the requested object. Here, we consider the average object sizes for calculating all delays, including the transmission delay and the transcoding delay. We apply a "sliding window" technique, for estimating access frequency, to make our model less sensitive to transient workload [17]. Specifically, the access frequency is estimated by $N/(t - t_N)$, where N is the number of accesses recorded, t is the current time, and t_N is the Nth most recently referenced time (the time of the oldest reference in the sliding window). N is set to 2 in the simulation.

4.2 Existing Algorithms

We include the following algorithms for evaluating our replacement algorithm proposed in Section 3.

- *LRU*: Least Recently Used (LRU) evicts the web object which was requested the least recently. The requested object is stored at each node through which the object passes. The cache purges one or more least recently requested objects to accommodate the new object if there is not enough room for it.

- *$LNC - R$ [15]*: Least Normalized Cost Replacement ($LNC - R$) is an algorithm that approximates the optimal cache replacement algorithm. It selects for replacement the least profitable documents. The profit function is defined as $profit(O_i) = (c_i \cdot f_i)/s_i$, where c_i is the average delay to fetch document O_i to the cache, f_i is the total number of references to O_i, and s_i is the size of document O_i.

- *AE [7]*: Aggregate Effect (AE) is a cache replacement algorithm in transcoding proxies that explores the aggregate effect of caching multiple versions of the same multimedia object in the cache. It formulates a generalized profit function to evaluate the aggregate profit from caching multiple versions of the same multimedia object. When the requested object passes through each node, the cache will determine which version of that object should be stored at that node according to the generalized profit.

From the above introduction, we can see that AE is a cache replacement algorithm in transcoding proxies, which is designed for handling the transcoded data. Our algorithm can be viewed as an improvement of this algorithm as discussed in previous sections. For the other two algorithms, i.e., LRU and $LNC-R$, they are not designed for handling the transcoded dada. Our purpose to include LRU and $LNC-R$ in the simulation is to show that the general cache replacement algorithms cannot be directly applied to solve the cache replacement problem for multimedia object caching in which the transcoded data are included.

5 Performance Evaluation

In this section, we compare the performance results of our algorithm with those algorithms introduced in Section 4.2, in terms of several performance metrics. The performance metrics we used in our simulation include delay-saving ratio (DSR), defined as the fraction of communication and server delays which is saved by satisfying the references from the cache instead of the server; average access latency (AAT); request response ratio (RRR), defined as the ratio of the access latency of the target object to its size; and object hit ratio (OHR), defined as the ratio of the number of requests satisfied by the caches as a whole to the total number of requests. In the following figures, LRU, $LNC-R$, and AE denote the results for the algorithms introduced in Section 4.2, OA denotes the efficient algorithm proposed in Section 3.

5.1 Impact of Cache Size

In this experiment set, we compare the performance results of different algorithms across a wide range of cache sizes, from 0.04 percent to 15.0 percent.

The first experiment investigates DSR as a function of the relative cache size at each node and Figure 3 shows the simulation results. As presented in Figure 3, we can see that our replacement algorithm outperforms the others. The mean reason is that LRU and $LNC-R$ cannot be applied to handle the transcoded dada and AE removes the candidates from the cache according to the individual profit. The advantage of our algorithm is contributed by the aggregate profit of the removed objects, which is not the simple summation of the individual profit. Specifically, the mean improvements of DSR over LRU, $LNC-R$, and AE are 23.2 percent, 18.7 percent, and 16.4 percent, respectively.

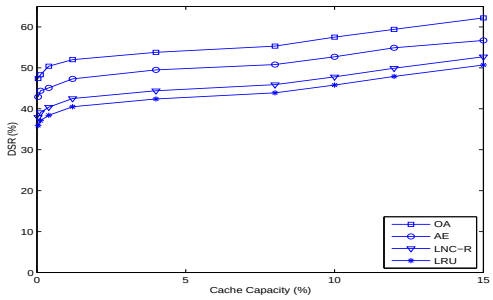

Figure 3: Experiments for DSR

Figure 4 shows the simulation results of AAL as a function of the relative cache size at each node; we describe the results of RRR as a function of the relative cache size at each node in Figure 5. Clearly, the lower the AAL or the RRR, the better the performance. As we can see, all algorithms provide steady performance improvement as the cache size increases. We can also see that OA improves both AAL and RRR compared to LRU, $LNC-R$, and AE. For AAL to achieve the same performance as OV, the other algorithms require 2 to 5 times as much cache size.

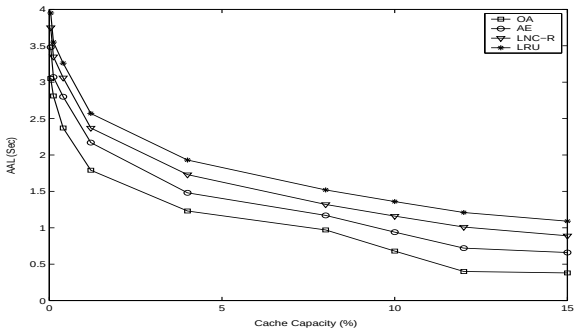

Figure 4: Experiments for AAT

Figure 6 shows the results of OHR as a function of the relative cache size for different algorithms. By computing the optimal versions to be cached, we can see that our replacement algorithm produces better results than the others, especially for smaller cache sizes. We can also see that OHR steadily improves as the relative cache size increases, which conforms to the fact that more requests will be satisfied by the caches as the cache size becomes larger.

5.2 Impact of Object Access Frequency

This experiment set examines the impact of object access frequency distribution on the performance results of the various algorithms. Figures 7, 8, and 9 show the performance results of DSR, RRR, and OHR respectively for the values of Zipf parameter α from 0.2 to 1.0.

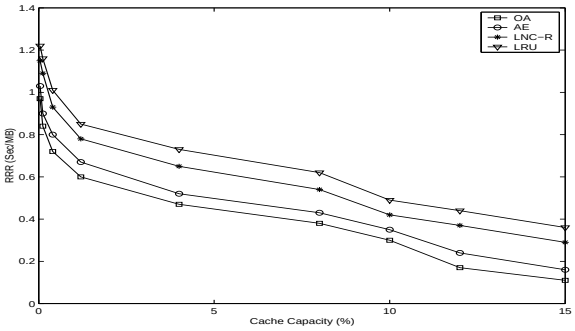

Figure 5: Experiments for RRR

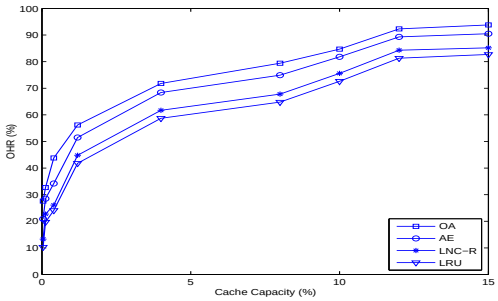

Figure 6: Experiments for OHR

We can see that OA consistently provides the best performance over a wide range of object access frequency distributions. As the parameter α increases, DSRs of all algorithms increase. Obviously, the more the object access frequency becomes concentrated, the more the overall hit ratio increases. This, in turn, leads to the increase of DSR. The advantage of our algorithm is contributed by taking the object access frequency to different versions of the same multimedia object into calculating the aggregate profit. Specially, OA reduces or improves DSR by 28.7 percent, 23.3 percent, and 19.6 percent compared to LRU, $LNC-R$, and AE, respectively; the default cache size used here (4 percent) is fairly large in the context of web caching, due to the large network under consideration.

6 Conclusion

In this paper, we addressed the problem of cache replacement for multimedia object caching. The objective is to minimize the total access cost by combining both transmission cost and transcoding cost. We also conducted a set of simulation experiments to study the performance of our algorithm by comparison with existing algorithms.

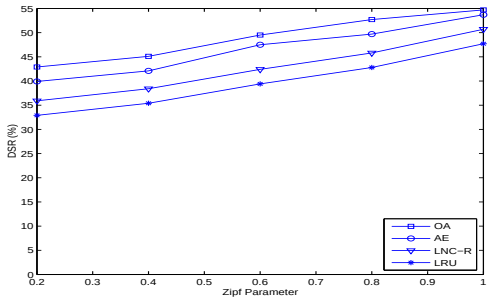

Figure 7: Experiments for DSR

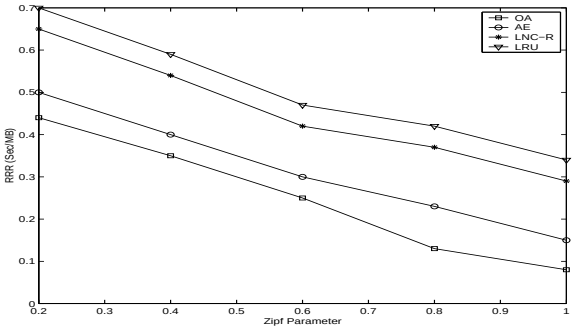

Figure 8: Experiments for RRR

References

[1] C. Aggarwal, J. L. Wolf, and P. S. Yu. *Caching on the World Wide Web.* IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 1, pp. 94-107, 1999.

[2] M. Arlitt and C. Williamson. *Trace-Driven Simulation of Document Caching Strategies for Internet Web Servers.* Simulation Journal, Vol. 68, No. 1, pp. 23-33, January 1997.

[3] A. Balamash and M. Krunz. *An Overview of Web Caching Replacement Algorithms.* IEEE Communications surveys, Vol. 6, No. 2, pp.44-56, 2004.

[4] P. Barford and M. Crovella. *Generating Representive Web Workloads for Network and Server Performance Evaluation.* Proc. ACM SIGMETRICS'98, pp. 151-160, 1998.

[5] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. *Web Caching and Zipf-like Distributions: Evidence and Implications.* Proc. IEEE INFOCOM'99, pp. 126-134, 1999.

[6] K. L. Calvert, M. B. Doar, and E. W. Zegura. *Modeling Internet Topology.* IEEE Communications Magazine, Vol. 35, No. 6, pp. 160-163, 1997.

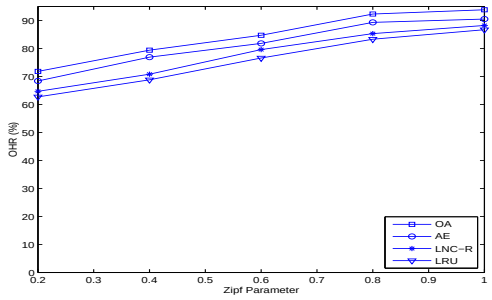

Figure 9: Experiments for OHR

[7] C. Chang and M. Chen. *On Exploring Aggregate Effect for Efficient Cache Replacement in Transcoding Proxies.* IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 6, pp. 611-624, June 2003.

[8] P. Cao and S. Irani. *Cost-Aware WWW Proxy Caching Algorithms.* Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems, pp. 193-206, December 1997.

[9] C. R. Cunha, A. Bestavros, and M. E. Crovella. *Characteristics of WWW Client-based Traces.* Technical Report TR-95-010, Boston University, April 1995.

[10] P. Cao and S. Irani. *Improving Proxy Cache Performance: Analysis of three Replacement Policies.* IEEE Internet Computing, Vol. 3, No. 6, pp. 44-50, 1999.

[11] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas. *Dynamic Adaptation in An Image Transcoding Proxy for Mobile Web Browsing.* IEEE Personal Communications, Vol. 5, No. 6, pp. 8-17, 1998.

[12] V. N. Padmanabhan and L. Qiu. *The Content and Access Dynamics of a Busy Site: Findings and Implications.* Proc. ACM SIGCOMM'00, pp.111-123, 2000.

[13] K. Psounis and B. Prabhakar. *Efficient Randomized Web-Cache Replacement Schemes Using Samples from Past Eviction-Times.* IEEE/ACM Transactions on Networking, Vol. 10, No. 4, pp. 441-454, August 2002.

[14] L. Rizzo and L. Vicisano. *Replacement Policies for a Proxy Cache.* IEEE/ACM Transactions on Networking, Vol. 8, No. 2, pp. 158-170, April 2000.

[15] P. Scheuermann, J. Shim, and R. Vingralek. *A Case for Delay-Conscious Caching of Web Documents.* Computer Networks and ISDN Systems, Vol. 29, nos. 8-13, pp. 997-1005, 1997.

[16] B. Shen, S.-J. Lee, and S. Basu. *Caching Strategies in Transcoding-Enabled Proxy Systems for Streaming Media Distribution Networks.* IEEE Transactions on Multimedia, Vol. 6, No. 2, pp. 375-386, April 2004.

[17] J. Shim, P. Scheuermann, and R. Vingralek. *Proxy Cache Algorithms: Design, Implementation, and Performance*. IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 4, pp. 549-562, 1999.

[18] D. Starobinski and D. Tse. *Probabilistic Methods for Web Caching*. Performance Evaluation , Vol. 46, nos. 2-3, pp. 125-137, October 2001.

[19] X. Tang, F. Zhang, and S. T. Chanson. *Streaming Media Caching Architectures for Transcoding Proxies*. Proc. of the 31st International Conference on Parallel Processing (ICPP), pp. 287-295, August 2002.

[20] A. Vetro, C. Christopoulos, and H. Sun. *Video Transcoding Architectures and Techniques: An Overview*. IEEE Signal Processing Magazine, Vol. 20, No. 2, pp. 18-29, 2003.

[21] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and F. A. Fox. *Removal Policies in Network Caches for World-Wide Web Documents*. Proc. of the ACM SIGCOMM'96 Conference, pp. 293-305, August 1996.

[22] Z. Xu, S. Sohoni, R. Min, and Y. Hu. *An Analysis of Cache Performance of Multimedia Applications*. IEEE Transactions on Computer, Vol. 53, No. 1, pp. 20-38, January 2004.