A Minimal Access Cost-Based Multimedia Object Replacement Algorithm

Keqiu Li[1,2], Takashi Nanya[2], and Wenyu Qu[3]
[1]School of Electronic and Information Engineering
Dalian University of Technology
No 2, Linggong Road, Dalian, 116024, China
[2] Research Center for Advanced Science and Technology
The University of Tokyo
4-6-1 Komaba, Meguro-ku, Tokyo, 153-8904, Japan
[3] College of Computer Science and Technology
Dalian Maritime University
No 1, Linghai Road, Dalian, 116026, China

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 cache replacement problem for multimedia object caching by exploring the aggregate effect of caching multiple versions of the same multimedia object. First, We present an optimal solution for calculating the minimal access cost of caching multiple versions of the same multimedia object. Second, based on this solution, we propose an effective cache replacement algorithm for multimedia object caching. Finally, we evaluate the performance of the proposed solution with a set of simulation experiments for various performance metrics over a wide range of system parameters.

Key words: Web caching, multimedia, cache replacement, transcoding, optimization, Internet.

1 Introduction

Web caching has been recognized as one of the effective solutions to alleviate web service bottlenecks, reduce traffic over the Internet and improve the scalability of the WWW systems. A survey of web caching schemes for the Internet can be found in [14]. There are a couple of elements that affect the performance of web caching, such as caching system architecture, proxy placement, cache routing, cache replacement, dynamic data caching, etc. In this paper, we study the cache replacement problem for multimedia objet caching, which is to decide the replacement candidates for accommodating a new object when the cache space is not enough under the condition that both web objects and multimedia objects are cached [1]. A number of cache replacement algorithms have been proposed in literature, which attempt to minimize various metrics, such as hit ratio, byte hit ratio, average latency, and total cost. An overview of web caching algorithms can be found in [1, 11]. However, most of these algorithms cannot be directly applied to solve the cache replacement problem for multimedia object caching since different versions of the same multimedia object cannot be simply viewed as different web objects, which has been shown in [6, 8].

In [6], the authors proposed an efficient cache replacement algorithm for transcoding proxies (AE for short) [2], which removes objects from the cache according to its generalized profit. 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. From the following example, we can see that this method is not optimal.

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

[1]In this paper, we consider a web object as a single-version multimedia object, while a multimedia object has at least two different versions.

[2]A transcoding proxy is proxy with the functionality of transcoding.

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

of each object are shown in Table 1, where $s(\cdot)$ and $p(\cdot)$ denote the size and the profit, respectively.

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 are the ones with the least profits, and the total size of them is enough for the new object. The profit we lose by removing these two objects is 32. We can see AE is not efficient in this case because the profit we lose 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 the above example, we can see that studying the cache replacement problem for multimedia object caching has significance on both theory and practice.

The main contributions of this paper are summarized as follows.

- We propose an optimal solution for calculating the minimal access cost of caching multiple versions of the same multimedia object.
- Based on this solution, we propose an effective cache replacement algorithm for multimedia object caching.
- We evaluate the performance of the proposed solution with a set of simulation experiments for various performance metrics over a wide range of system parameters.

The rest of this paper is organized as follows. Section 2 introduces some preliminary concepts of transcoding. In Sections 3, we present an optimal solution for the problem of multimedia object replacement and its analysis. The simulation model and performance evaluation are described in Sections 4 and 5, respectively. Section 6 summarizes our work and concludes the paper.

2 Object Transcoding

Transcoding is a technology that is used to transform a multimedia object from one version to another, frequently through trading off object fidelity for size [5, 7, 12, 13]. The relationship among different versions of a multimedia object can be expressed by a weighted transcoding graph [6]. An example of such a graph is shown in Figure 1. In Figure 1, we can see that the original version A_1 can

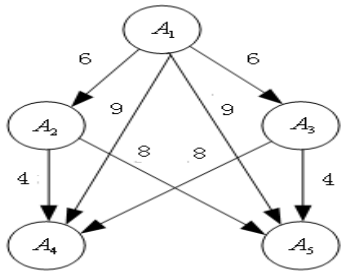

Figure 1. A Weighted Transcoding Graph

be transcoded to each of the less detailed versions A_2, A_3, A_4, and A_5. It should be noted that not every A_i can be transcoded to A_j since it is possible that A_i does not contain enough content information for the from A_i to A_j. The transcoding cost from A_i to A_j of a multimedia object is denoted by $t(A_i, A_j)$, which is defined as the delay for transcoding one version to another for the same multimedia object. Obviously, $t(A_i, A_i) = 0$. If a version cannot be transcoded from another version, we consider the transcoding cost as infinity. If version A_j can be transcoded from version A_i through version A_k with $i < k < j$, then $t(A_i, A_j) \leq t(A_i, A_k) + t(A_k, A_j)$ (triangularity property) because version A_k is always an option if the transcoding cost $t(A_i, A_j)$ is too large. In this paper, the transcoding graph is a linear array and the transcoding cost between any two adjacent versions is constant (T), i.e., $t(A_i, A_j) = \sum_{k=i}^{j-1} t(A_k, A_{k+1}) = (j-i)^+ T$, where $x^+ = x$

if $x \geq 0$ else $x^+ = \infty$. Our analysis can be easily extended for the case in which the transcoding graph is a general transcoding graph as defined in the beginning of this section.

3 An Effective Cache Replacement Algorithm for Multimedia Object Caching

For the cache replacement problem, only the node with a cache and the server are considered [3]. Assume that a multimedia object O has m different versions, denoted by $A_1, A_2, \cdots, A_m$, where A_1 is the most detailed version and A_m is the least detailed version. For version A_i ($1 \leq i \leq m$), we associate the link between the cache and the server a nonnegative cost L_i, which is defined as the cost of sending a request for version A_i and the relevant response over the link. So we have $L_1 \geq L_2 \geq \cdots \geq L_m$. In this paper, we define this cost as the transmission cost. Obviously, Let f_i be access frequency of version A_i from the cache.

Before presenting the cache replacement algorithm for multimedia object caching, we calculate the access cost of caching k versions of a multimedia object (AC-k problem for short), where $1 \leq k \leq m$. In [9], we studied this problem under the assumption that $L_1 = L_2 = \cdots = L_m = L$.

First, we begin by computing the access cost of caching only one version A_k at the cache with $1 \leq k \leq m$. Now we discuss how a request for a specific version is served. Obviously, the requests for the versions that cannot be transcoded from A_i shall be served by the server, while some of the requests for A_i with $i \geq k$, depending on the transcoding cost and the transmission cost, will be taken care of by transcoding from version A_k. Therefore, the total access cost of caching only version A_k at the cache is computed as follows:

$$C(A_k) = \sum_{i=1}^{k-1} f_i L_i + \sum_{i=k}^{m} f_i \min\{(i-k)T, L_i\} \qquad (1)$$

Since only version A_k is cached, there must exist an integer δ such that $(\delta - i)T < L_i$ and $(\delta - i - 1)T \geq L_{i-1}$. Therefore, δ is such a parameter that the request for version A_i will be served by the cache if $0 < i < \delta$, and the request for version A_i will be served by the server if $i \geq \delta$.

Based on Equation (1), $C(A_k)$ can be further defined as follows:

$$C(A_k) = \begin{cases} \sum_{i=1}^{k-1} f_i L_i + \sum_{i=k}^{k+\delta-1} f_i (i-k)T \\ + \sum_{i=k+\delta}^{m} f_i L_i \ (k+\delta \leq m) \\ \sum_{i=1}^{k-1} f_i L_i + \sum_{i=k}^{m} f_i (i-k)T \ (k+\delta > m) \end{cases} \qquad (2)$$

It is easy to see that $C(A_1)$ can be calculated in $O(m)$ time. Thus, each $C(A_2), C(A_3), \cdots, C(A_m)$ can be done in constant time; Therefore, the AC-1 problem can be solved in $O(m)$ time. Therefore, based on the cost function as given in Equation (1), the AC-1 problem of caching only one version (i.e., $n = 1$) from the set $\{A_1, A_2, \cdots, A_m\}$ can be solved in $O(m)$ time.

The second step is to extend the above solution to compute the optimal solution for caching two versions, A_{k_1} and A_{k_2}, at the same time at node v.

Suppose that A_{k_1} and A_{k_2} are the two optimal versions to be cached. The key observation here is that A_{k_1} is also an optimal solution for the problem with $\{A_1, A_2, \cdots, A_{k_2-1}\}$ if $k_1 < k_2$, because the requests for $\{A_{k_2}, A_{k_2+1}, \cdots, A_m\}$ can only be served by A_{k_2}. Regarding to this observation, we have the following lemma.

Lemma 1 *Assume that A_{b_p} and A_{b_q} are the optimal solutions for the problem of caching only one version from the set of $\{A_1, A_2, \cdots, A_{p-1}\}$ and $\{A_1, A_2, \cdots, A_{q-1}\}$ 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 A_{b_p} is the optimal version to be cached, we have $C_{1,p}(A_{b_p}) < C_{1,p}(A_{b_{p+1}})$, where $C_{1,p}(A_i)$ denotes the access cost of caching A_i for the MOP problem with $\{A_1, A_2, \cdots, A_{p-1}\}$. From the definition of the access cost function $C_{1,p}$ as given in Equation (1), adding A_p to the set $\{A_1, A_2, \cdots, A_{p-1}\}$ will increase both $C_{1,p}(A_{b_p})$ and $C_{1,p}(A_{b_{p+1}})$ by $f_p \min\{(p-b_p)T, L_p\}$ and $f_p \min\{(p-b_{p+1})T, L_p\}$ respectively. The increase to $C_{1,p}(A_{b_{p+1}})$ is no less than that to $C_{1,p}(A_{b_p})$ because $b_p > b_{p+1}$. So we have $C_{1,p+1}(A_{b_p}) < C_{1,p+1}(A_{b_{p+1}})$, which contradicts the fact that $C_{1,p+1}(A_{b_{p+1}})$ is the minimum access cost of caching $A_{b_{p+1}}$ for the problem with $\{A_1, A_2, \cdots, A_{p-1}, A_p\}$. Hence the lemma is proven.

Based on Lemma 3, we can see that the feasible range of the optimal solution for the problem with $\{A_1, A_2, \cdots, A_q\}$ can be reduced if the optimal version for the problem with $\{A_1, A_2, \cdots, A_p\}$ has been obtained. So is the

[3] Here, the server can be a cache, which holds a version of a multimedia object.

other case when the optimal solution for the problem with $\{A_1, A_2, \cdots, A_q\}$ is known, the feasible range of the optimal solution for the problem with $\{A_1, A_2, \cdots, A_p\}$ is also reduced. Therefore, we can find A_{b_p} and compute $C_{1,p}(A_p)$ by divide and conquer.

Let $D^{(k)}_{p,q}$ denote the minimum access cost of caching k versions for the AC-1 problem with $q-p$ versions, i.e., $A_p, A_{p+1}, \cdots, A_{q-1}$, where $1 \le p < q \le m$. Thus, $D^{(1)}_{1,p} = C_{1,p}(A_{b_p})$ and $D^{(1)}_{1,m+1} = \min\limits_{1\le k\le m}\{C_{1,m+1}(A_k)\}$. Based on Lemma 3, we have the following theorem on the time complexity of computing $D^{(1)}_{1,p}$ for $1 < p \le m$.

Theorem 1 *All the m AC-1 problems for $\{A_1, A_2, \cdots, A_p\}$ where $1 \le p \le m$, i.e., $D^{(1)}_{1,p}$ for $1 < p \le m$, can be computed in $O(m \log m)$ time.*

Proof Assume that there exists an integer θ such that $m = 2^\theta$, then we can compute $D^{(1)}_{1,\frac{1}{2}m}$ in $O(m)$ time. Assume that $A_{b_{\frac{m}{2}}}$ is the optimal solution for the problem of caching only one version with $\{A_1, A_2, \cdots, A_{\frac{m}{2}-1}\}$, then we can find the optimal solution for the problem of caching only one version for $\{A_1, A_2, \cdots, A_{\frac{m}{4}}\}$ in $O(m)$ time. Similarly, $D^{(1)}_{1,\frac{3m}{4}}$ can also be computed by solving the problem of caching only one version with $\{A_1, A_2, \cdots, A_{\frac{3m}{4}-1}\}$. As we have already computed $C_{1,\frac{m}{2}}(A_y)$ where $y = \min(b_{\frac{m}{2}}, \frac{m}{2} - 1)$, we can base on this result to compute $C_{1,\frac{3m}{4}}(A_y)$ for $\{A_1, A_2, \cdots, A_{\frac{3m}{4}-1}\}$ (by adding at most $\frac{m}{4}$ terms to $C_{1,\frac{m}{2}}(A_{\frac{m}{2}-1})$. We then compute $C_{1,\frac{3m}{4}}(A_y), C_{1,\frac{3m}{4}}(A_{y+1}), \cdots, C_{1,\frac{3m}{4}}(A_{\frac{3m}{4}-1})$ in at most $O(\frac{3m}{4} - y)$ time. So it takes at most $O(m)$ time to compute $D^{(1)}_{1,\frac{m}{4}}$ and $D^{(1)}_{1,\frac{3m}{4}}$. According to the similar decomposition, $D^{(1)}_{1,\frac{m}{8}}$, $D^{(1)}_{1,\frac{3m}{8}}$, $D^{(1)}_{1,\frac{5m}{8}}$, and $D^{(1)}_{1,\frac{7m}{8}}$ can all be solved in $O(m)$ time. After repeating this process $\log m$ times, we can finish computing $D^{(1)}_{1,p}$ for $1 < p \le m$. 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 $1 < p \le m$, 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., $A_1, A_2, \cdots, A_{p-1}$. In particular, $D^{(1)}_{1,m+1} = \min\limits_{1\le k\le m}\{C_{1,m+1}(A_k)\}$.

- *Step* 2: Evaluate $\overline{D}_p$ for $2 \le p \le m$, where $\overline{D}_p$ is the access cost for versions $A_p, A_{p+1}, \cdots, A_m$ if A_p is cached at node v. $\overline{D}_p$ is defined as follows:

$$\overline{D}_p = \begin{cases} \sum\limits_{i=p}^{p+\delta-1} f_i(i-p)T + \sum\limits_{i=p+\delta}^{m} f_i L_i & \text{if } p+\delta \le m \\ \sum\limits_{i=p}^{m} f_i(i-p)T & \text{if } p+\delta > m \end{cases}$$

- *Step* 3: Compute $D^{(2)}_{1,m}$, where $D^{(2)}_{1,m}$ is the minimum access cost of caching two versions for the problem with $\{A_1, A_2, \cdots, A_m\}$. $D^{(2)}_{1,m}$ is calculated as follows:

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

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

After we have calculated $D^{(1)}_{1,p}$ for $1 \le p \le m$ in *Step* 1, we can obtain $D^{(2)}_{1,p}$ for all $2 \le p \le m$ in another $O(m \log m)$ 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., $A_1, A_2, \cdots, A_{p-1}$. The main idea is similar to Lemma 3 in the finding of $D^{(1)}_{1,p}$. Assume that $A_{b_{p_1}}$ and $A_{b_{p_2}}$ with $1 \le b_{p_1} < b_{p_2} < p$ are the two optimal versions cached in node v for $A_1, A_2, \cdots, A_{p-1}$ to achieve the optimal access cost $D^{(2)}_{1,p}$. Similarly, $A_{b_{q_1}}$ and $A_{b_{q_2}}$ with $1 \le b_{q_1} < b_{q_2} < q$ are the two optimal versions cached in node v for $A_1, A_2, \cdots, A_{q-1}$ to achieve the optimal access cost $D^{(2)}_{1,q}$. We can show with a similar argument with Lemma 3 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 3, the two optimal solutions in $D^{(2)}_{1,\frac{m}{2}}$ can be found in $O(m)$ time after knowing the optimal versions of $D^{(1)}_{1,p}$ for $1 < p \le m$; then $D^{(2)}_{1,\frac{m}{4}}$ and $D^{(2)}_{1,\frac{3m}{4}}$ in another $O(m)$ time; then $D^{(2)}_{2,\frac{m}{8}}$, $D^{(2)}_{1,\frac{3m}{8}}$,$D^{(2)}_{1,\frac{5m}{8}}$, and $D^{(2)}_{1,\frac{7m}{8}}$ in another $O(m)$ time until $D^{(2)}_{1,p}$ for $2 < p \le m$ are found after $\log m$ times. Therefore, the minimum access cost of caching three versions, denoted by $D^{(3)}_{1,m}$, can be computed similarly, i.e., $D^{(3)}_{1,m} = \min\limits_{3\le p\le m}\{D^{(2)}_{1,p} + \overline{D}_p)\}$, with at most $O(m \log m)$ time. Using the same idea, we can solve the problem of caching K versions in $O(Km \log m)$ time.

Let $D^{(K)}_{1,m}$ denote the minimum access cost of caching K versions from m versions, i.e., $A_1, A_2, \cdots, A_m$, then it is easy to show that $D^{(K)}_{1,m}$ can be computed in $O(Km \log m)$ time.

Based on the above analysis, we can present a cache replacement algorithm for multimedia object caching such that the total size of the removed objects is greater than the size of the new object and the generalized cost loss of the removed objects is minimized. Here, the generalized cost loss of the removed objects is defined as the total access cost when they are not cached divided by their total size. Suppose there are n different objects cached and the size of a new object to be cached is s_{New}, then we should find a subset of objects $O^* \subseteq O$ that satisfies the following conditions:

1. $\sum_{o_{i,j} \in O^*} s_{i,j} \geq s_{New}$

2. $C(O^*)/S(O^*) \leq C(O^{'})/S(O^{'}), \ \forall \ O^{'} \subseteq O$, where $O^* = \{o^*_{1,1}, o^*_{1,2}, \cdots, o^*_{1,\alpha_1}, o^*_{2,1}, o^*_{2,2}, \cdots, o^*_{2,\alpha_2}, \cdots, o^*_{\beta,1}, o^*_{\beta,2}, \cdots, o^*_{\beta,\alpha_\beta}\}$ is the set of objects to be removed, $O = \{o_{1,1}, o_{1,2}, \cdots, o_{1,m_1}, o_{2,1}, o_{2,2}, \cdots, o_{2,m_2}, \cdots, o_{n,1}, o_{n,2}, \cdots, o_{n,m_n}\}$ is the set of objects cached, $C(O^*) = \sum_{i=1}^{\beta} C(o^*_{i,1}, o^*_{i,2}, \cdots, o^*_{i,\alpha_i})$, and $S(O^*)$ is the total size of all the objects in O^*. $C(O^{'})$ and $S(O^{'})$ can be similarly defined. Obviously, (1) is to make enough room for the new object, and (2) is to evict those objects whose generalized aggregate cost saving is minimized.

Before we present the algorithm, we give some notations used in the algorithm. Let $R^*(i,k)$ denote the minimal generalized aggregate cost loss of removing k versions of object i and $R^*(k)$ denote the minimal generalized cost loss of the k objects removed. We can see that the k objects removed can be k versions of a multimedia object or different versions of different multimedia objects, i.e., k can be decomposed as $k = k_1 + k_2 + \cdots + k_n$, where $0 \leq k_i < k$ is the number of versions of object i that are in the set of the k objects removed. It can be easily proved that there are k different such combinations in all. Therefore, we have $R^*(k) = \min\{R^*(1,k), R^*(2,k), \cdots, R^*(n,k), \min_{k=k_1+k_2+\cdots+k_n}\{R^*(k_1)+R^*(k_2)+\cdots+R^*(k_n)\}\}$. We denote the set of all the objects that achieves $R^*(k)$ by $O^*(k)$ and their total size is $S^*(k)$. 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_{New} is the object to be cached, and its size is s_{New}. The algorithm is shown in Table 2.

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

Theorem 2 *The time complexity of the proposed algorithm is $O(n^2 \log n)$, where n is the total number of different objects cached.*

Proof The running time of the proposed algorithm mainly depends on Steps 4, 6, and 8. Based on the previous analysis, it is easy to see that the running time of Step 6 is $O(\sum_{i=1}^{n} l \cdot m_i \log m_i)$ since there are n different objects and the running time for object i for removing l versions of object i is $O(l \cdot m_i \log m_i)$, where m_i is the number of versions of object i. The running time for Step 8 is $O((n+l)\log(n+l))$ because we should order all $n+l$ items to find the minimal one among them. Thus, the total running time for the proposed algorithm (Step 4)is $O(\sum_{l=1}^{k}[(l+n)log(l+n)+\sum_{i=1}^{n} l \cdot m_i \log m_i]) = O(n^2 \log n)$ since $k \leq n$ and $m_i \ll n$. Hence, the theorem is proven.

Table 2. The Cache replacement Algorithm

1	INSERT o_{New} INTO C
2	$k = 0$
3	$S^*(k) = 0$
4	WHILE $S_c - S_u - S^*(k) < s_{New}$ DO
5	$k = k + 1$
6	FOR $i = 1$ TO n DO
7	CALCULATE $R^*(i,k)$
8	CALCULATE $R^*(k)$

From Theorem 3, we know that the time complexity of the proposed algorithm 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 arrives at 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 the proposed algorithm is $O(n \log n)$, which is the same as that of the algorithm proposed in [6]. 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 aggregate cost savings for them whenever we insert or evict an object into or from the cache. Such operations are, in general, very costly. Here, we can save calculated results for later computation, which will save a lot of computations.

4 Simulation Model

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 [2, 3, 4, 6].

The network topology was randomly generated by the Tier program [4]. Experiments for many topologies with various parameters were conducted and the performance of our solution was found to be insensitive to topology

Table 3. Parameters Used in Simulation

Parameter	Value
Delay of Fetching Objects	Exponential Distribution ($\theta = 1.5Sec$)
Web Object Size Distribution	Pareto Distribution ($\mu = 6KB$)
Web Object Access Frequency	Zipf-Like Distribution ($\alpha = 0.7$)
Average Request Rate	$U(1, 9)$ requests
Transcoding Cost	$50KB/Sec$

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 3, which were chosen from the open literature and are considered to be reasonable.

We also assume that each multimedia object has five versions. The sizes of each version are assumed to be 100 percent, 80 percent, 60 percent, 40 percent, and 20 percent of the original object size. The transcoding delay is determined as the quotient of the object size to the transcoding rate.

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

- LRU: Least Recently Used (LRU) evicts the web object which was requested the least recently.
- $LNC - R$ *[10]*: Least Normalized Cost Replacement ($LNC - R$) removes the least profitable documents.
- AE *[6]*: Aggregate Effect (AE) selects the object with the least generalized profit for replacement.

5 Performance Evaluation

In this section, we compare the performance results of our solution with those solutions introduced in Section 4, 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; 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 solutions introduced in Section 4, OA denotes the optimal solution proposed in Section 3. Due to space limitation, we will not give the detailed description on the performance improvement of our solution

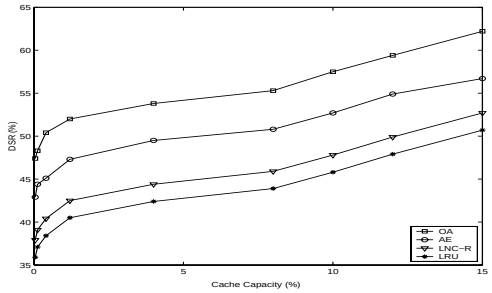

Figure 2. Experiments for DSR

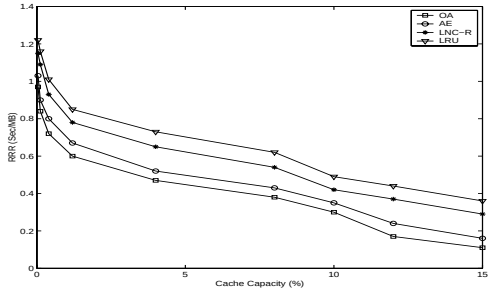

Figure 3. Experiments for RRR

6 Conclusions

In this paper, we addressed the cache replacement problem for multimedia object caching and proposed an effective algorithm for soling this problem. A set of simulation experiments were conducted to study the performance of our solution. The simulation results show that our solution improves network performance compared with existing solutions.

References

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

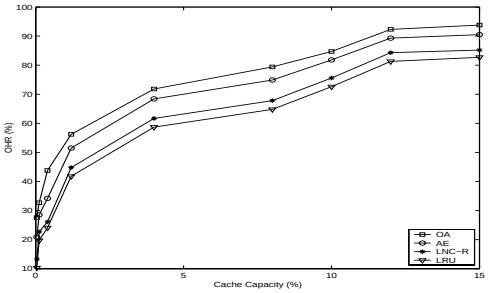

Figure 4. Experiments for OHR

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

[3] 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.

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

[5] S. Chandra, C. Ellis, and A. Vahdat. *Application-Level Differentiated Multimedia Web Services Using Quality Aware Transcoding*. IEEE Journal on Selected Areas in Communications, Vol. 18, No. 12, pp. 2544-2565, 2000.

[6] 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.

[7] 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.

[8] K. Li, H. Shen, and F. Chin. *Placement Solutions for Multiple Versions of a Multimedia Object*. Proc. of The 8th IEEE International Symposium on Object-oriented Real-time distributed Computing, pp. 224-231, May 2005.

[9] K. Li, H. Shen, K. Tajima, and L. Huang. *An Effective Cache Replacement Algorithm for Transcoding Proxy Caching*. Journal of Supercomputing, Vol. 35, No. 2, pp. 165-184, 2006.

[10] P. Scheuermann, J. Shim, and R. Ving ralek. *A Case for Delay-Conscious Caching of Web Documents*. Computer Networks and ISDN Systems, Vol. 29, No. 8-13, pp. 997-1005, 1997.

[11] S. Podlipnig and L. Boszormenyi. *A Survey of Web cache Replaceemnt Strategies*. ACM Computing Syrveys, Vol. 35, No. 4, pp. 374-398, December 2003.

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

[13] 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.

[14] J. Wang. *A Survey of Web Caching Schemes for the Internet*. ACM Computer Communication Review, Vol. 29, No. 5, pp. 36-46, 1999.