MAXIMIZING THE PROFIT FOR CACHE REPLACEMENT IN A TRANSCODING PROXY

Hao-Ping Hung and Ming-Syan Chen

Graduate Institute of Communication Engineering
National Taiwan University
E-mail: mschen@cc.ee.ntu.edu.tw, hphung@arbor.ee.ntu.edu.tw

ABSTRACT

Recent technology advances in multimedia communication have ushered in a new era of personal communication. Users can ubiquitously access the Internet via various mobile devices. For the moible devices featured with lower-bandwidth network connectivity, transcoding can be used to reduce the object size by lowering the quality of a mutimedia object. In this paper, we focus on the cache replacement policy in a transcoding proxy, which is a proxy server responsible for transcoding the object and reducing the network traffic. Based on the architecture in prior works, we propose a Maximum Profit Replacement algorithm, abbreviated as MPR. MPR performs cache replacement according to the content in the *caching candidate set*, which is generated by the concept of *dynamic programming*. Experimental results show that the the proposed MPR outperforms the prior scheme in terms of the *cache hit ratio.*

Keywords — Transcoding Proxy, Cache Replacement Policy.

1. INTRODUCTION

The technology advance in network has accelerated the development of multimedia applications over the wired and wireless communication. To alleviate network congestion and to reduce latency and workload on multimedia servers, multimedia proxy has been proposed to cache popular contents. Caching the data objects can relieve the bandwidth demand on the external network, and reduce the average time to load a remote data object to local side. Since the effectiveness of a proxy server depends largely on cache replacement policy, various approaches are proposed in recent years [1][2][4][6][9]. Subject to the application context, the cache replacement policy may differ from one situation to another, such as caching scheme in transcoding proxy [3], multiserver cache replacement [10], and cache replacement for wireless data access [8][11].

In this paper, we focus on the cache replacement policy in a transcoding proxy. A transcoding proxy is a proxy capable of converting a multimedia object from one form to another, which trades object fidelity for size. Explicitly, we formulate the caching strategy as a *0-1 knapsack problem* [5] and show that algorithm AE (standing for Aggregate Effect) in [3] can be viewed as a *greedy* algorithm. Note that the *greedy* algorithm only performs well in *fractional knapsack problem*. In order to improve the performance of AE, we propose a Maximum Profit Replacement algorithm, abbreviated as MPR. Given a specific cache size constraint, we first calculate the caching priority by considering the fetching delay, the reference rate, and the item size of the objects. By using the concept of *dynamic programming* [7], the *caching candidate set* D_H, which contains the objects with high caching priority, will be determined. Extended from the conventional two running parameters in solving *0-1 knapsack problem*, the proposed procedure DP (standing for *Dynamic Programming*) comprises three parameters to retain the information of the cache size, the object id and the version id. According to the content in D_H, MPR performs cache replacement to maximize the value of *profit*. Algorithm MPR can be divided into two phases. The first phase performs when the proxy has sufficient space. In this phase, once the object is queried, it will be cached to increase the *profit* for future access. The second phase performs when the proxy has insufficient space. In this phase, the cache replacement is determined according to the priority of the requested object. We also show that algorithm MPR is of polynomial time complexity. To verify the effectiveness of MPR, several experiments are conducted. The experimental results show that the proposed MPR outperforms AE in maximizing the *cache hit ratio*.

This paper is outlined as follows. Preliminaries are given in Section 2. In Section 3, we describe the design of MPR. The experimental results are shown in Section 4. Finally, this paper concludes in Section 5.

2. NOTATION AND DEFINITIONS

A list of the symbols used is given in Table 1. We assume that each object can be represented in n versions. The original version of object i is denoted as $o_{i,1}$, whereas the least detailed version which cannot be transcoded any more is denoted as $o_{i,n}$.

Definition 1: The weighted transcoding graph, G_i, is a directed graph with weight function ω_i. G_i depicts the transcoding relationship among transcodable versions of object i. Each vertex $v \in V[G_i]$ represents a transcodable version of object i. Version u of object i, i.e., $o_{i,u}$ is transcodable to version v, i.e., $o_{i,v}$ iff there is a directed edge $(u, v) \in$

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

Table 1. Description of the symbols

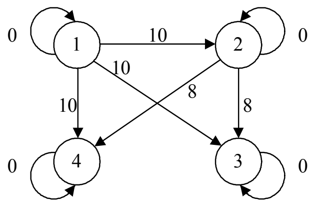


Fig. 1. The weighted transcoding graph

$E[G_i]$. The transcoding cost from version u to v is given by $\omega_i(u, v)$ which is the weight of the edge from u to v. Figure 1 illustrates the example of a weighted transcoding graph.
Definition 2: $PF(o_{i,j})$ is defined as the *singular profit* from caching $o_{i,j}$ while no other version of object i is cached.

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

Definition 3: $PF(o_{i,j1}, o_{i,j2}, ..., o_{i,jk})$ is defined as the *aggregate profit* from caching $o_{i,j1}, o_{i,j2}, ..., o_{i,jk}$ at the same time.

$$\begin{aligned} &PF(o_{i,j1}, o_{i,j2}, ..., o_{i,jk}) \\ = &\sum_{v\in V[G'_i]} \sum_{(v,x)\in E[G'_i]} r_{i,x} * (d_i + \omega_i(1, x) - \omega_i(v, x)), \end{aligned}$$

where G'_i is the subgraph which minimizes the aggregate transcoding cost when caching a certain set of versions of the object.
Definition 4: $PF(o_{i,j}|o_{i,j1}, o_{i,j2}, ..., o_{i,jk})$ is defined as the *marginal profit* from caching $o_{i,j}$, given $o_{i,j1}, o_{i,j2}, ..., o_{i,jk}$ are already cached where $j \neq j1, j2, ..., jk$.

$$\begin{aligned} PF(o_{i,j}|o_{i,j1},\dots, o_{i,jk}) &= PF(o_{i,j}, o_{i,j1}, o_{i,j2},\dots, o_{i,jk}) \\ &\quad -PF(o_{i,j1}, o_{i,j2},\dots, o_{i,jk}). \end{aligned}$$

3. MAXIMUM PROFIT REPLACEMENT ALGORITHM

Consider a database D which represents the collection of all possible queried data objects. A data object $o_{i,j} \in D$, which represents the j-th version of the i-th object in D, contains two attributes: *profit* $p_{i,j}$ and object size $s_{i,j}$. Note that $p_{i,j} = PF(o_{i,j})$ if no other version of object i is cached, and $p_{i,j} = PF(o_{i,j}|o_{i,j1},\dots, o_{i,jk})$ if there are other versions $o_{i,j1},\dots, o_{i,jk}$ cached.
Definition 5: The *caching candidate set*, D_H, is defined as a subset of D, where each object in D_H has high priority to be cached.
Definition 6: The *total profit* of D_H, denoted by P_H, is defined as the summation of the *profit* of all data objects, including the original and transcoded ones, in D_H. The *total size* of D_H, denoted by S_H, is defined as the summation of the object size of all data objects in D_H.
Definition 7: The *generalized profit*, $g_{i,j}$, of the object $o_{i,j}$ is defined as $g_{i,j} = p_{i,j}/s_{i,j}$.

3.1. Description of Procedure DP

Since the goal of this paper is to select the cached objects to maximize the total profit, we first determine the priority of the objects. All of the objects in D is divided into two parts: high priority and low priority. The objects of high priority are contained in D_H. The problem of determining D_H can be formulated as a *0-1 knapsack problem* [5], in which P_H and S_H can be viewed as the entire *value* and entire *weight* of the items that a thief can carry. As we know, the *0-1 knapsack problem* can be optimally solved by the concept of *dynamic programming* [7]. Therefore, the *caching candidate set* D_H can be optimally determined by the following procedure DP (standing for *Dynamic Programming*).

Procedure DP
Input: the database D, and the cache size constraint Z_c. Note that $|D| = m \times n$, where there are m kinds of data, and each data object has n versions.
Output: the *caching candidate set* D_H

```
begin
1.  Create a two-dimensional $(m \times n + 1) \times (Z_c + 1)$ array
    $c[][]$ and set each element $c[i][j]$ to be 0.
2.  for $i$:=1 to $m$ do
3.    for $j$:=1 to $n$ do
4.      for $z$:=1 to $Z_c$ do
5.        if $z_i \le z$
6.          if $p_{i,j} + c[(i-1)*n+j-1][z-z_i]$
               $> c[(i-1)*n+j-1][z]$
7.            $c[(i-1)*n+j][z]$
               $\leftarrow p_i + c[(i-1)*n+j-1][z-z_i]$
8.          else
9.            $c[(i-1)*n+j][z]$
               $\leftarrow c[(i-1)*n+j-1][z]$
10.       else
11.         $c[(i-1)*n+j][z]$
             $\leftarrow c[(i-1)*n+j-1][z]$
12.Determine $D_H$ by tracing the elements in $c[][]$
end
```

3.2. Description of Algorithm MPR

The complexity of procedure DP is $O(|D| \times Z_c)$, where Z_c is the cache size constraint. Therefore, D_H can be deter-

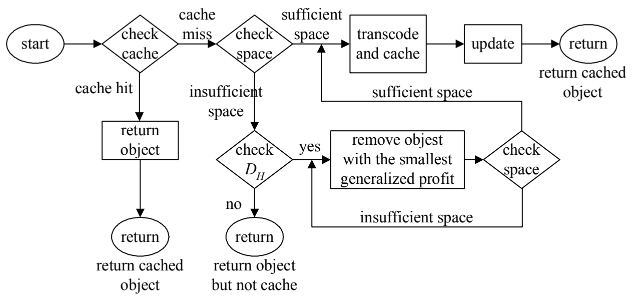


Fig. 2. Flowchart of algorithm MPR

mined in polynomial time. Algorithm MPR is outlined as follows:

Algorithm MPR

Figure 2 depicts the flowchart of algorithm MPR. The detail of the algorithm is summarized as follows.

1. For each incoming query, check the cached objects. If *cache hit* occurs, return the cached object.

2. If *cache miss* occurs, check the requested object size and the available space in the cache. If the available space of the cache is larger than or equal to the requested object size, and the requested object can be transcoded from some current object in the cache, perform transcode operation and put the transcoded object in the cache. If the available size of the cache is larger than or equal to the requested object size, but the requested object cannot be transcoded from any object in the cache, download the original version from the remote server, transcode it into the required version, and put it in the cache. Finally, update the *profit* of the associated objects and D_H.

3. If the available space of the cache is smaller than the requested object size, check the *caching candidate set* D_H. If the requested object belongs to D_H, remove the object with the smallest *generalized profit* until the available space of the cache is larger than or equal to the size of the requested object. Put the requested object into the cache in the way the same as in 2, and update the *profit* of the associated objects and D_H.

Algorithm MPR can be divided into two phases. The first phase performs when the proxy has sufficient space to cache the requested item. In this phase, once the object is queried, it will be cached to increase the *profit* for future access. That is, the proxy will cache as many objects as it can. The second phase performs when the proxy has insufficient space to cache the requested object. In the phase, the cache replacement is determined according to the priority of the requested object. The replacement policy of MPR is to keep the object with high priority and remove the object with low priority. Since the *caching candidate set* D_H contains all the objects with high priority, if the requested object belongs to D_H, replacement will occur to keep the requested data. We also use the *generalized profit*, $g_{i,j}$, to decide the order of removing objects with low priority. Note that an object with low *generalized profit* tends to have large item size or low *profit*. Therefore, the object with the smallest *generalized profit* in the cache will be removed first. The removal procedure will be executed iteratively until the proxy has enough space to cache the requested object with high priority.

Parameters	Values
Number of objects	500
Skewness parameter (θ)	$0.6 \sim 1.4$
Diversity parameter (Φ)	$100 \sim 500$
Cache capacity	10%
Channel bandwidth	1 unit/sec
Transcoding rate	20 unit/sec

Table 2. Simulation parameters

4. PERFORMANCE EVALUATION

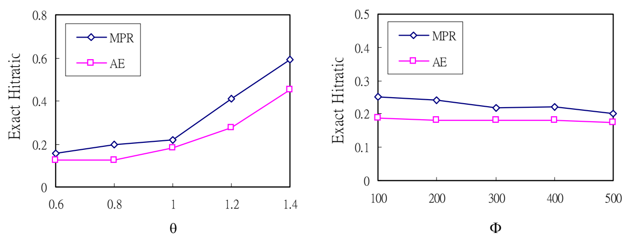
To verify the effectiveness of MPR, several experiments are conducted in this section. In Section 4.1, we describe the simulation model. In Section 4.2, the experimental results will be shown. In Section 4.3, the remarks of the experiments will be given.

4.1. Simulation Model

The simulation model is designed to reflect the system environment of the transcoding proxy in Section 2. Table 2 summaries the definitions for some primary simulation parameters. In the client model, mobile devices can be partitioned into five classes. That is, each data item can be transcoded to five different versions by the transcoding proxy to satisfy the users' requirements. The sizes of the five versions are set to be 100%, 80%, 60%, 40%, 20% of the original object size. A more detailed version can be transcoded to a less detailed one. Also, the distribution of these five mobile appliances is modeled as a device vector of <15%, 20%, 30%, 20%, 15%>. As for the transcoding proxy model, we assume that there are 500 different objects, and choose the cache capacity to be $0.1 * (\sum\text{object size})$. The reference rate of each object is generated by Zipf distribution $r_i = (\frac{1}{i})^\theta \sum_{j=1}^{N}(\frac{1}{j})$, where θ is viewed as a *skewness parameter*. The size of each object is represented by the uniform distribution between $(0, \Phi]$ units, where we define Φ as the *diversity parameter*.

4.2. Experimental Results

During the experiments, unlike the various metrics used in [3], we choose the tightest one, the *exact hit ratio*, to verify the performance of MPR. The *exact hit ratio* is defined as the fraction of requests which are satisfied by the exact versions of the objects cached. This metric is also motivated by the fact that we usually intend to provide an exact version to



(a) when the skewness parameter varied (b) when the diversity parameter varied

Fig. 3. The exact cache hit ratio v.s. (a) the skewness parameter and (b) the diversity parameter

users (rather than an overqualified one) for effective bandwidth use. Figure 3 shows the *exact hit ratio* of MPR and AE as the value of θ and Φ varies. Since the comparison between AE and other schemes has been made in [3], we focus on comparing the performance between MPR and AE in this paper. In Figure 3(a), the *exact hit ratio* increases as the value of θ increases. This phenomenon can be explained as follows. The *profit* is composed of two components: the reference rate and the delay saving. The increase of the skewness will make the *profit* dominated by the reference rate. Also, the increasing skewness enhances the locality of the object requests. Therefore, the *exact hit ratio* increases as the value of θ increases. On the other hand, Figure 3(b) depicts that the *exact hit ratio* will slightly decrease as the value of Φ increases. Since the increase of Φ will granulate the object size, it will be more likely that the cache space may not be utilized very thoroughly, which affects the performance of algorithm MPR.

4.3. Remarks

From the above two figures, we observe that the proposed MPR consistently outperforms algorithm AE [3]. The reason is as follows. In Section 3, given the *profit,* object size and available cache space, the cache replacement in transcoding proxy can be modeled as a *0-1 knapsack problem*. In algorithm AE, a heap is built according to the value of the *generalized profit* of each object. If the proxy has insufficient space to cache all of the objects in the heap, the object with the smallest *generalized profit* is excluded. That is, algorithm AE always caches the objects with higher *generalized profit*. Such behavior in algorithm AE can be viewed as a *greedy algorithm*. Note that the *greedy algorithm* achieves optimal solution only in the *fractional knapsack problem* [5]. In solving the *0-1 knapsack problem*, since algorithm MPR performs based on the *caching candidate set*, which is generated according to the concept of *dynamic programming*, MPR hence consistently outperforms AE.

5. CONCLUSION

In this paper, we explore the cache replacement policy in a multimedia transcoding proxy. Based on the architecture proposed in [3], we propose an effective caching scheme MPR to improve the existing algorithm AE. MPR performs cache replacement according to the content of the *caching candidate set*, which stores the objects with high caching priority. The experimental results show that the proposed MPR outperforms AE in terms of the *exact cache hit ratio*, which is the tightest of all performance metrics used in [3].

Acknowledgements

The work was supported in part by the National Science Council of Taiwan, R.O.C., under Contracts NSC93-2752-E-002-006-PAE.

6. REFERENCES

[1] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching Proxies: Limitations and Potentials. In *4th International World-Wide-Web Conference*, 1995.
[2] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout. Measurements of a Distributed File System. In *the 13th Symposium on Operating System Principles*, 1991.
[3] C.-Y. Chang and M.-S. Chen. On Exploring Aggregate Effect for Efficient Cache Replacement in Transcoding Proxies. *IEEE Transaction on Parallel and Distributed Systems*, 14(6), 2003.
[4] H. Chou and D. DeWitt. An Evaluation of buffer Management Strategies for Relational Database Systems. In *11th VLDB Conf.*, 1985.
[5] T. C. et al. Introduction to Algorithms. *McGraw Hill.*
[6] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K Page Replacement Algorithm for Database Disk Buffering. In *Int. Conf. Management of Data*, 1993.
[7] M. Sniedovich. Dynamic Programming. *M. Dekker*, 1992.
[8] J. Xu, Q. Hu, W.-C. Lee, and D. L. Lee. Performance Evaluation of an Optimal Cache Replacement Policy for Wireless Data Dissemination. *IEEE Transaction on Knowledge and Data Engineering*, 16(1), 2004.
[9] J. Yoon, S. L. Min, and Y. Cho. Buffer Cache Management: Predicting the Future from the Past. In *International Symposium on Parallel Architectures, Algorithms and Networks, 2002. I-SPAN '02. Proceedings*, 2002.
[10] Q. Zhang, Z. Xiang, W. Zhu, and L. Gao. Cost-based Cache Replacement and Server Selection For Multimedia Proxy Across Wireless Internet. *IEEE Transaction on Multimedia*, 6(4), 2004.
[11] B. Zheng, J. Xu, and D. Lee. Cache Invalidation and Replacement Strategies for Location-Dependent Data in Mobile Environments. *IEEE Transaction on Computers*, 51(10), 2002.