EFFICIENT SEGMENT-BASED VIDEO TRANSCODING PROXY FOR MOBILE MULTIMEDIA SERVICES

Kuei-Chung Chang, Ren-Yo Wu, Tien-Fu Chen

Department of Computer Science and Information Engineering
National Chung Cheng University, Taiwan, R.O.C.

ABSTRACT

To support various bandwidth requirements for mobile multimedia services for future heterogeneous mobile environments, a transcoding video proxy is usually necessary to provide adapting video streams to mobile clients by not only transcoding videos to meet different needs on demand, but also caching them for later use. Traditional proxy technology is not applicable to a video proxy because it is less cost-effective to cache the complete videos to fit all kind of clients in the proxy server. In this paper, we propose the *object relation graph* (ORG) to manage the static relationships between video versions, and utilize the *cached object relation tree* (CORT) and replacement algorithm to manage video segments cached in the proxy dynamically. Experimental results show that the proposed algorithm significantly outperforms companion schemes in terms of the byte hit ratios.
.

1. INTRODUCTION

Requests for multimedia contents over the web have become more and more popular. A proxy performs an important service provider between content servers and clients on reducing response time and network traffic. With mobile technology grows quickly, more mobile devices have the capability of playing low-quality videos. So, transcoding video proxy would become more and more important.

The function of a video transcoding proxy is the same with a general media transcoding proxy (e.g. image transcoding, document distiller) to help mobile clients to reduce the file size and to fit to mobile device capabilities. Although we can extend the traditional caching model to support streaming media, it usually does not take advantages of streaming characteristics. For example, it is too large to cache the entire video objects. In addition, the transmission of streaming objects needs to be rate regulated and these timing constraints need to be considered in the design of a proxy for streaming media.

Streaming video playback usually face some problems, such as insufficiency of server-proxy bandwidth, transcoding delay, the proxy-client bandwidth for the proxy to receive video on time, and limited capability of mobile devices. In order to solve these problems, it can cache objects in the proxy either for transcoding to other format on the fly or for future use directly. The proxy does not cache entire videos, but some important video versions that can be transcoded to other versions with low quality to reduce the network traffic from video server and save the storage of the proxy. Several caching strategies for streaming media [3, 5, 6, 8] have been proposed in recent years by caching a portion of a video file at the proxy. In particular, caching an initial pre-fix of a video [4, 7, 9] has a number of advantages including shielding clients from delays and jitters on the path from the server to the proxy, while reducing traffic along the path.

In this paper, we examine the trade-off of caching frequently used video objects for on-the-fly transcoding video objects to fit the network, CPU and storage requirements in the proxy. The motivation for this work is mainly due to the new emerging factors in the environment of transcoding proxies. First, we should consider the reference rates to different versions of a video object separately because the distribution of them could affect the caching decision. Second, the traditional cache replacement algorithms could make a wrong replacement decision without considering the transcoding delay. We use the *object relation graph* to manage the static relationships between video versions, and utilize the *cached object relation tree* to manage video objects cached in the proxy dynamically. The objective of the proxy is to store the most valued video objects that have the most related transcoding profits to improve the proxy byte hit ratio and reduce the retransmission times from video servers.

The rest of this paper is organized as follows: In Section 2 we present our proposed method of the transcoding proxy including the cache management and the replacement policy. Section 3 shows experimental results of our method. Finally, Section 4 concludes the paper.

2. OUR PROPOSED METHOD

In this section, we describe the system architecture of the transcoding proxy, and discuss the cache management and replacement policy.

2.1. System environment

Figure 1 is the architecture of our proposed transcoding proxy server that consists of a request manager and a cache manager. The request manager is in charge of finding out the proper video format for different kinds of client devices. Moreover, the request manager also controls the transcoder to choice a suitable video version based on user preferences, client device capacities, and network characteristics. The cache manager manages video versions and cached objects. It decides which object should be cached, and which object should be replaced when the cache space is full. It also manages transcoded video objects and original video objects that can be transcoded to other video versions. The video buffer stores video objects transmitted from the server when the proxy cache missed. The transcoded buffer is a space that stores transcoded video objects ready to send to clients. A media file contains multiple equal-sized blocks that are the smallest unit of transfer. A segment contains multiple blocks, and the cache admission and replacement policies will attach different caching values to each segment. The user request will be accepted while the network and CPU resource are available. Otherwise, the request is blocked.

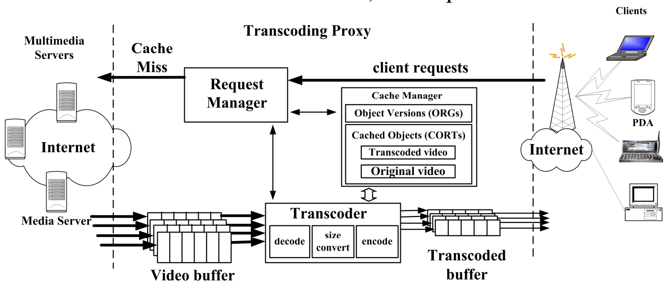


Fig. 1. Transcoding proxy architecture

2.2. Cache management

2.2.1. *Video object versions*

When the proxy transcodes a video version to another video version with lower quality, it must reference more detailed resolution videos to perform the transformation. We depict the dependent relationships of transcoding video versions by the *object relation graph (ORG)*, shown as Figure 2. We define the object relation graph as following:

Definition 2.1 (Object Relation Graph) *The object relation graph, G, is a directed graph with a transcoding delay weight function w. G depicts the static transcoding relationships among transcodable versions of a video.*

1. *For each vertex v ∈ V[G] represents a transcoded version of a video with different quality, and it keeps the information of the frame rate and the frame size.*
2. *For each directed edge e(u, v) ∈ E[G] represents the transcoding delay weight. Let e(u, v) = (transcoding time from v to u / playback time of u) ≈ {(F(u)×R(u)) / ER$_{vu}$ }, where the F(u) represents the frame size of u, the R(u) represents the frame rate of u and the ER$_{vu}$ represents the encoding rate (e.g. [12]) from v to u. The transcoding delay weights of dotted edge are less or equal than one. It represents that version u can be transcoded from version v in reasonable transcoding time.*

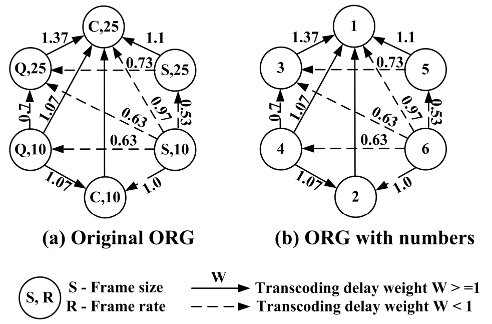


Fig. 2. Object relation graph with transcoding delay

Figure 2(a) shows the example of the ORG, and we represent video versions as numbers instead of detailed video information, shown as Figure 2(b). The CPU time required for encoding is proportional to the image resolution, and the encoding time dominates the overall transcoding time. Let version(q) represents the version number of the object with quality q. If version(u) ≤ version(v), the version u can be transcoded to version v. For example, in Figure 2(b), objects 2, 3, 4, 5, 6 dependent on object 1 and they can be transcoded from object 1. Object 4 can also be transcoded from objects 1, 2 and 3, and object 6 can be transcoded from objects 1, 2, 3, 4, 5.

A vertex with at least one dotted link connecting to other vertices represents that this video version could be transcoded from other video versions. From the object relation graph, we can find the static dependent relationships between video versions. We can use the dependent relationships to find out which video versions can be transcoded from other video versions cached in the proxy. If the video version can be transcoded to several video versions in a reasonable transcoding time, it is a valuable video version. For example, the directed edge $e(V\ (Q,\ 25),\ V\ (C,\ 25)) = 1.37$ indicates that transforming format CIF with 25 fps video to QCIF with 25 fps video requires transcoding time more than playback time 1.37 times, shown as Figure 2(a). Therefore, node $V\ (Q,\ 25)$ is not worth transcoded, and we should cache it in the proxy.

2.2.2. *Cache admission policy*

The goal of caching video objects in a proxy is to reduce network latency of clients and to reduce the load of video servers. However, video objects are always larger than traditional text and image objects. With limited storage constraints, the video proxy usually cannot cache all versions of all videos. The proxy only caches the requested and the valuable segments, and we use a tree structure to store these cached segments. The *object relation graph*, which shows static relation information of different video versions, it cannot reveal the actual information of objects cached in the proxy. Therefore, we use the *cached object relation tree* (CORT) to represents the dynamic information of cached video objects in the proxy and defined as following:

Definition 2.2 (Cached Object Relation Tree) *The cached object relation tree, Ti, is a tree structure with segment id, popularity and video version information. Ti depicts the actual relationships among a related group of the cached objects of video i at run time.*

1. *For each node v ∈ V[Ti] represents a cached segment of video i. It keeps the video format, popularity, caching profit and segment information. We use the exponential-sized segmentation [10] to save the media objects. The root node represents the conceptual node that represents video i. It is designed just for video indexing, and it can be kept in a hash table for searching quickly.*
2. *For each edge e ∈ E [Ti] represents the transcoding delay weight previously defined in object relation graph. The edge from root to the node u of next layer is ((F (u) × R (u)) / Server-proxy network bandwidth).*

Figure 3 shows an example of the cached object relation tree. Node a, b, c, and d are full version and cached in the proxy. Node e, f, g, h and i may be transcoded from their parent objects. When we insert, remove or retrieve a node, we will update the tree and the node information.

2.3. Replacement policy

When the proxy space is full, we have to select the useless objects as victims. In this section, we propose a CPU-time constrained replacement policy. The popularity is decided by the access rate which is estimated as $f = 1 / (T - T')$, where T is the current time and T' is the last access time. A value of f close to one represents that the object is requested recently. The more popular video object has higher priority to be cached. However, it is not enough in our method. The main decisions are calculated from the object transcoding delay and the popularity. If an object is cached, it saves its transcoding time and may save other relative object's transcoding time. We define the function of cache object profit to each object as following:

Definition 2.3 (CPUCacheProfit) *Let $O_{i,j}$ denotes the version j of object i. Let $f_{i,j}$ denotes the reference rate of $O_{i,j}$. We define the CPUCacheProfit($O_{i,j}$) function for the profit of caching $O_{i,j}$ as following:*

$$CPUCacheProfit(O_{i,j}) = f_{i,j} \times \{ e_g(j, P_j) +$$
$$[\sum(e_g(M, P_j) - e_g(M, j)) - \sum(e_g(C_j, P_j) - e_g(C_j, j))] \},$$

where e_g represents the edge in the ORG, M represents the vertex that has a link entering vertex j in the ORG, P_j represents the parent vertex of j in the CORT and C_j represents the child vertices of j in the CORT.

The first part of the equation represents the gain which caching object version j can save the cost transcoded from other object versions. The second part of the equation represents the gain which caching object version j can save the transocding time of other relative object versions that can be transcoded from version j.

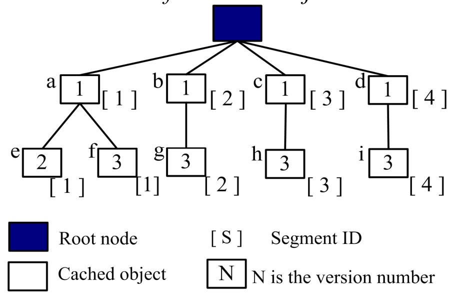
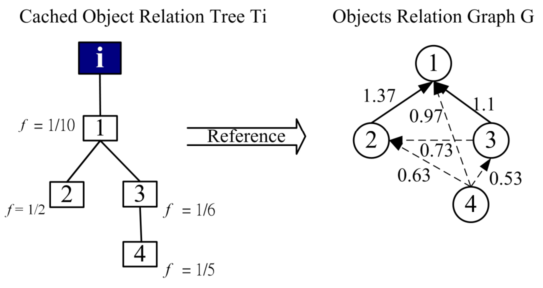


Fig. 3. Example of the cached object relation tree

Figure 4 shows an example of the CPUCacheProfit calculation. The cost of caching node 3 would save the transcoding time for itself and node 4. The cost of caching node 4 can only save the transcoding time for itself. In this example, we can see that the cache profit of node 2 is more than that of node 3. It means that cache node 2 is more useful than node 3. Figure 5 is the CPU-time constrained replacement algorithm.



$$\text{CacheProfit}(2) = (1/2)\{\, \underline{1.37} + (\underline{[(1.1\text{-}0.73) + (0.97\text{-}0.63)}]\text{-}\ \underline{[0]}) \} = 1.04$$
$$\text{CacheProfit}(3) = (1/6)\underline{\{\ 1.1\ } + (\underline{[(0.97\text{-}0.53)}]\ \text{-}\ \underline{[(0.97\text{-}0.53)}]) \} = 0.183$$
$$\text{CacheProfit}(4) = (1/5)\underline{\{\ 0.53} + \underline{(0}) \} = 0.106$$

Fig. 4. CacheProfit calculation example

```
Algorithm CPU-time_Constrained_Algorithm(v)
  v : the requested version object
  If (the request if not blocked) {
    While (there is not enough free space for v) {
      look for idle object u in the cache with the lowest
          CPUCacheProfit(u);
      if (CPUCacheProfit(u) ≧ CPUCacheProfit(v)) {
        return;
      } else {
        remove u;
        update the CPUCacheProfit of u, child(u) and parent(u)
      }
    }
    cache v;
    update the CPUCacheProfit of v and parent(v)
  }
```

Fig. 5. CPU constrained replacement algorithm

3. EXPERIMENTAL RESULTS

In our experiment environments, we use the exponentially sized segment approach [10] to define the storage unit of the video object. The popularity of video objects follows Zipf-like distribution $Zipf(x, M)$ [11], where M is total distinct video titles. The distribution is given by $p_i = (c / i^{1-x})$ for each $i \in 1, \ldots M$, where $c = 1/[\sum_{i=1}^{x} 1/i^{1-x}]$, is a normalization constant. We set $M = 2000$ and $x = 0.2$ for video popularity.

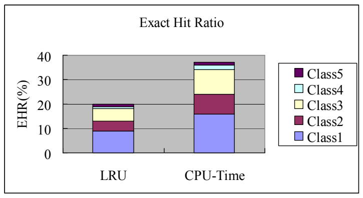
The interarrival time between two consecutive access sequences is modeled by an exponential distribution with the mean value of 0.5 seconds. The client's silent time between two successive requests within an access sequence is modeled by a Pareto distribution with the mean value of 60 seconds. The video size is distributed between 1000B and 3000B, where B is the block size. The playing time for a block B is assumed 1.8 seconds. In other words, the playing time for a video is between 30 minutes and 90 minutes. The cache size is expressed in terms of number of media blocks, and the default cache size is 400,000 blocks.

For the client model, as in [1], we classify the mobile appliances into five classes. The distribution of these five classes of mobile clients is modeled as a device vector including 15%, 20%, 30%, 20% and 15%. The size of five versions of each object are assumed to be 100%, 80%, 60%, 40% and 20% of the original object size (frame rate x frame size).

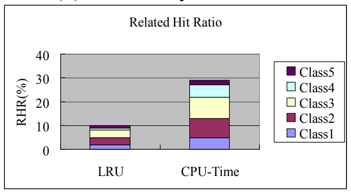
The main performance metric in our experiments is *byte hit ratio*. The byte hit ratio represents the ratio of the total bytes accessed from cached objects directly (Exact Hit) or transcoding from relative objects (Related Hit) over the total bytes of objects requested. In Figure 6, we can see that the related hit ratio of the CPU-time constrained policy is much higher than that of LRU (Least Recently Used) [2] in Figure 6(b). This is because that LRU does not consider the transcoding relationships between cached objects to decide the right victims. Our cache management will replace the less related objects that cannot be transcoded to other versions.

4. CONCLUSIONS

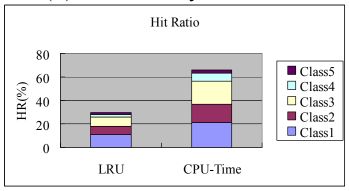
In this paper, we construct an interesting model caching different versions of video objects to serve variant types of client devices. We consider that the lower resolution video objects can be transcoded from other appropriate related objects to speedup the response time to mobile clients in heterogeneous network environments. In the experimental results, we can see that our replacement policy will take the transcoding time as the key decision consider to select the right objects as victims. In the future, we will cooperate with client buffer to dynamic adjust the transmission rate according to the client, the server and the network characteristics.



(a) Exact byte hit ratio



(b) Related byte hit ratio



(c) Total byte hit ratio

Fig. 6. Hit ratios of five classes of client.

5. REFERENCES

[1] V. Cardellini, P.-S. Yu, and Y.-W. Huang., "Collaborative proxy system for distributed web content transcoding," In Proceedings of ACM International Conference on Information and Knowledge Management, pp. 520–527, 2000.

[2] P. J. Denning., "The working set model for program behavior," Communications of the ACM, 11(5):323–333, 1968.

[3] D. Eager, J. Almedia, and M. Vernon., "A hybrid caching strategy for streaming media files," In Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking, January 2001.

[4] Stephane Gruber, Jennifer Rexford, and Andrea Basso., "Protocol considerations for a prefix-caching proxy for multimedia streams," In Proceedings of the 9th International World Wide Web Conference, May 2000.

[5] Katherine Guo, Sanjoy Paul, Hui Zhang, and T.S. Eugene Ng. Markus Hofmann., "Caching techniques for streaming multimedia over the internet," In Technique Report, Bell Laboratories, April 1999.

[6] Keith W. Ross, Martin Reisslein, and Felix Hartanto., "Interactive video streaming with proxy servers," In Proceedings of International Workshop on Intelligent Multimedia Computing and Networking, 2000.

[7] Subhabrata Sen, Jennifer Rexford, and Don Towsley., "Proxy prefix caching for multimedia streams," In Proceedings of IEEE INFOCOM, 1999.

[8] Brian Smith and Soam Acharya., "Middleman: A video caching proxy server," In Proceedings of Workshop on Network and Operating System Support for Digital Audio and Video, June 2000.

[9] B. Wang, S. Sen, M. Adler, and D. Towsley., "Optimal proxy cache allocation for efficient streaming media distribution," In Proceedings of IEEE INFOCOM, 2002.

[10] Kun-Lung Wu, Philip S. Yu, and Joel L. Wolf., "Segment-based proxy caching of multimedia streams," In Proceedings of the 10th International Conference World Wide Web, pp. 36–44, 2001.

[11] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-Like Distributions:Evidence and implications,", Proc. IEEE INFOCOM, Mar. 1999

[12] Warabino, A. Ota, S. Morikawa, D. Ohashi, M. Nakamura, H. Iwashita and H. Watanabe, "Video transcoding proxy for 3G wireless mobile Internet access," IEEE Communications Magazine, 38(5):66-71, October 2000