META-CACHING AND META-TRANSCODING FOR SERVER-SIDE SERVICE PROXY

Bo Shen

Hewlett-Packard Laboratories, 1501 Page Mill Road 1181, Palo Alto, CA 94304
boshen@hpl.hp.com

ABSTRACT

This paper proposes novel concepts of meta-caching and meta-transcoding which indicate that intermediate results (metadata) of service (e.g., video transcoding) sessions are cached and subsequent identical service requests are served through transcoding from the cached metadata. The proposed methods provide a foundation to achieving superior storage and computation resource balance at service proxies. It is primarily useful in reducing the computing load of service proxies at the server side. The paper further identifies the appropriate metadata and its characteristics for three types of video transcoding tasks. Simulations based on these characteristics validate that the proposed methods achieve superior computing load reduction on transcoding service proxies.

1. INTRODUCTION

For streaming media delivery in heterogeneous network environments, service proxies with transcoding capability can be employed for real-time content adaptation upon request arrival. Services of this kind are often computing intensive tasks, which have motivated developments of fast video transcoding algorithms in the compressed domain (e.g., [1]). Another solution is to use caching schemes. For example, the final results from a transcoding session can be cached so that future identical requests can be served without another transcoding session. However, traditional caching schemes (e.g., [2]) have been designed mainly for server-to-proxy traffic reduction. They are less efficient in alleviating the problem of intensive computing load on the proxy. Moreover, the design of server-side service proxies is more concerned with the reduction of average computing load rather than the reduction of server-to-proxy traffic.

The traditional caching scheme reduces the computing load on the service proxy by caching the final results of service sessions, which requires a large cache space considering the large volumes of video objects. This is especially true when considering heterogeneous video delivery in which case the final results of transcoded video objects may have many different variations. In addition, this scheme leads to the fact that cached objects impose no computing load but missed objects imposes full computing load. That is, the storage resource may become scarce while plenty of computing resource is available. The exhaustion of the storage resource may result in more cache misses and the computing load may increase uncontrollably. This situation calls for the design of new caching schemes focusing on caching for computation.

In this paper, we explore the tradeoffs between computing and storage requirements so that the two different types of resources on a proxy system can be used in a balanced fashion. The design goal is to reduce the average computing load on the proxy but maintain the CPU utilization so that storage resources can be used more efficiently. This goal is achieved by identifying, in service processes, intermediate steps at which intermediate results (metadata) can be cached and the final result can be easily produced from the intermediate results. The proxy can then choose to cache the intermediate results that may occupy less cache space. Effectively, metadata related to more sessions can be cached. While the computing load per cached session is increased compared with the situation when final results are cached, the benefit of caching for more sessions may outweigh the penalty we pay for the slight increase of CPU load per service session.

This paper uses video transcoding as an example service to discuss the advantages of the meta-caching and the meta-transcoding methods. Section 2 proposes the meta-caching system and its operating principles. Section 3 introduces a suite of three meta-transcoding algorithms. Simulation results and performance evaluations are provided in Section 4. We then conclude and address future work in Section 5.

2. META-CACHING PRINCIPLES

Figure 1 illustrates a service proxy system with meta-transcoding capability. The proxy sits in between an origin content server that provides the input stream, and a group of heterogeneous clients that consume the output stream. The transcoding proxy is connected with the content server through high-speed links. The contents are transcoded according to client requests before transmitted through the Internet. This setup is often found in many server-side proxy architectures. The cache space on the proxy is partitioned into two sections. Regular cache is used to store the final results of transcoding sessions. Meta cache is used to store the metadata which is defined as the intermediate result produced during a transcoding session. Given the metadata in the cache, meta-transcoding is defined as the continued transcoding process based on the metadata that is produced and cached from a prior identical session. In an overly simplified example, if a transcoding service is to obtain \( x \) by adding \( a, b \) and \( c \) the proxy can choose to cache the intermediate result of \( a+b \) so that \( x \) is obtained by one less addition in the next identical service session. Note that the above scenario is favorable only if the intermediate result \( (a+b) \) occupies less cache space than the final result \( x \).
If nothing is cached, e.g., there is no cache at the transcoding proxy; we call it the “no-caching” method. In this case, the service proxy always starts a new session upon request arrival and the per-session computing load is 100%. In another extreme, if the system caches the final results, we call it the “final-caching” method. In this case, no computing is required for a future identical session assuming unlimited cache space. The proposed meta-caching method operates in the range between the no-caching and final-caching scenarios. The resource balancer makes the decision based on the cache space and computing cycles available. In general, if the system has sufficient idle computing cycles but not enough cache space, metadata instead of final results can be cached. Conversely, final results instead of metadata can be cached.

Ideally, the selected metadata should occupy less cache space while alleviating bottlenecks of transcoding processes. Considering a transcoding proxy that performs transcoding on compressed MPEG video objects, the transcoding flow may produce metadata at five different levels. The tradeoff between the storage and computing requirements incurred by the caching of the metadata at these levels is roughly depicted in Figure 2.

If the proxy caches the sequence level metadata, less (compared to no-caching) computing resource is required for a future identical session. The sequence level metadata includes header information, total frame number, etc. These data describe overall characteristics of the whole video object and usually occupy small amount of storage. When the next identical request arrives, the sequence level metadata can be used to carry out the transcoding session with lighter load on the CPU. For example, if the header information is available, the new transcoding session does not need to parse the initial part of the video and the initialization process is simpler. However, the bulk of the transcoding process is still to be carried out, therefore the saving on computing is minimum.

Caching of metadata at the picture and macroblock level requires more storage resource than caching of sequence level metadata. However, since the final results can be produced much easier from the metadata at these levels, less computing resource is required for a future identical session. We discuss the caching of the metadata at these levels and the corresponding meta-transcoding processes in more detail in Section 3.

Caching metadata does not necessarily achieve performance gain. For example, if the proxy caches the pixel level metadata (e.g., the reconstructed pixel frames), excessive storage resource is required since it is uncompressed information. In addition, the cached pixel frames need to be encoded into final bit stream and thus additional computing resource is still required. In general, it makes no sense to employ the meta-caching method when the storage-computing tradeoff character of the selected metadata that is closer to the final results, in the extreme case, cache the final result. The fine granularity thus achieved enables a more efficient utilization of different types of resources available at the service proxy.

3. META-CACHING AND META-TRANSCODING ALGORITHMS

We discuss meta-caching and meta-transcoding algorithms for three typical types of video transcoding tasks in this section.

3.1. Meta-caching for temporal rate reduction

This is typically achieved by frame dropping. For example, a MPEG video with a GOP (group of picture) structure of IBPBBBPBB can drop the B-pictures to achieve a 1/3 frame rate reduction. Note that the bottleneck of the transcoding is at the parsing of the input bit streams. The purpose of the parsing is to find out the coding type and byte offset of picture.

For this type of transcoding, we propose to cache the picture level metadata, specifically, picture type (1 byte – can be optimized) and picture byte offset (4 bytes – cover objects size up to 4 GB) information of the original video in the meta-cache. Therefore, a total of 5 bytes (40 bits) of cache space per picture is required. Given the frame rate of the original video at \( f \) frames per second (fps), the cache-filling rate of the metadata is \( R_t = 40f \) bits per second (bps). Note that a 30 fps video at any bit rate can now be cached at about 1 Kbps, the cache space required is reduced significantly. On the other hand, since the type of a coded picture and its associated byte offset are now readily available in the meta-cache, the meta transcoder can easily service a repeated transcoding request without parsing. The computation load on the meta transcoder is slightly bigger compared with final-caching in which case no computing is required. Frame rate reduction usually leads to bit rate reduction. If the target bit rate of the output stream is smaller than \( R_t \), the
3.2 Meta-caching for bit rate reduction

Here, we consider specifically the case when frame rate and spatial resolution are maintained, only the bit rate of the output stream is reduced. This is typically achieved by requantization of the transform coefficients. Figure 4 illustrates the processing flow of bit rate reduction of a MPEG video. We find that caching of the requantization scale factor (Mq) achieves a good tradeoff between storage and computing resource utilization.

![Figure 3. Bit Rate Reduction with Meta-Caching.](image)

The metadata is the macroblock (of size 16x16 for MPEG) level information as shown in Figure 2. The cached Mq table speeds up future similar transcoding session since no rate control is necessary. In general, given output video stream with pixel resolution \( w \times h \) and frame rate \( f \) fps, there is a total of \( w \times h \times f / (16 \times 16) \) macroblocks per second. Consider using 1 byte (8 bits) for caching of Mq for each macroblock, the cache-filling rate of the metadata is \( R_s = w \times h \times f / 32 \) bps. If the target bit rate of the output stream is smaller than \( R_s \), the final-caching method should be employed instead of meta-caching.

3.3 Meta-caching for spatial resolution reduction

This is achieved by reducing the resolution of each frame and coding the result at a lower bit rate. Figure 4 illustrates the processing flow.

![Figure 4. Spatial Resolution Reduction with Meta-Caching.](image)

In this transcoding process, there are many computing intensive modules. For video encoding, the computing of motion vector (MV) is most computing-intensive. If the system can cache the result of the MV Generator, a subsequent request to the same service can be served with less computation. Assuming no field-encode for output videos, macroblocks in an inter-picture may be associated with 0 to 2 motion vectors. Therefore, this type of metadata requires 4 bytes (32 bits) for each macroblock. In general, given an output video stream with pixel resolution \( w \times h \), frame rate \( f \) fps and GOP size of \( n \), total number of inter-pictures is \( f \times (n-1) / n \) per second. The cache-filling rate of the metadata is \( R_n = w \times h \times f \times (n-1) / 8n \) bps. If the target bit rate of the output stream is smaller than \( R_n \), the final-caching should be employed instead of meta-caching.

Resolution reduction transcoding usually leads to bit rate reduction. Similar to the bit rate reduction transcoding, the result from the rate control module can also be cached to avoid estimation of new quantization factor in future sessions. Same analysis applies as discussed in Section 3.2.

### 4. RESULTS AND PERFORMANCE EVALUATION

The full transcoder and meta transcoder are implemented in C with no special optimizations. The implementation considers the transcoding operations in the compressed domain, i.e., compressed information are reused as much as possible [1]. Through transcoding trail runs on HP X4000 workstation with 2 GHz Intel Xeon CPU, we compare the CPU time used by full transcoding and meta transcoding for bit rate reduction on an MPEG test sequence with spatial resolution 352x240. The original video contains I-, P- and B-pictures and is coded at 1 Mbps, it is transcoded to 256 Kbps. Figure 5 plots the CPU time used in transcoding of each frame. It shows that meta transcoding requires 50% less computation than the full transcoding. Normalizing the computing load from 0 (no transcoding) to 1 (full-transcoding) unit, a meta-transcoding session costs a fractional computing unit. Considering output streams as web video clips of resolution 352x240 with 25 fps, we obtain the average computing load of meta transcoding per session by trial runs for a group of MPEG sequences. Table 1 shows the average load per session and cache-filling rate for three types of meta-transcoding tasks as discussed in Section 3. The magnitude of the load in the table may vary for different implementations and running platforms since it represents the relative load with respect to full transcoding load per session. Cache-filling rate for meta-caching are calculated based on the analysis in Section 3 (GOP size is 12).

To evaluate the performance of meta-caching algorithms, we simulate the behavior of a meta-caching system using video access pattern generated as follows. The popularity of a total of 1000 original video objects follows a Zipf-like distribution with the skew factor \( \alpha =0.47 \) [3]. The length of these video objects varies from 5 to 15 minutes. All video objects are transcoded before delivered to the client and the proxy caches the transcoded final results or metadata. The simulation contains 20000 accesses arriving through a random Poisson process with an average interval of 4 seconds. Assuming all requests ask for a transcoding to generate output at an average bit rate of 512 Kbps, there is a total of 67 GB of unique transcoded objects. The simulator uses LRU as the cache replacement policy. Based on cache capacities in various percentages of the unique transcoded object size, we study the aggregated computation load of the transcoding proxy.
We first evaluate the computing load reduction effect for bit rate reduction transcoding. Figure 6 plots the aggregated computing load at each access arrival when the cache capacity is 5 GB (1000 accesses in the middle of the access sequence are shown). Compared with the no-caching scenario (Total in figure), the final-caching (Final in figure) method reduces the computing load by 20%. Meta-caching (Meta-B in figure) method achieves a superior 55% reduction.

Figure 7 shows the average computing load with various cache capacities for all three meta-transcoding tasks. The computation and storage requirements of the meta-transcoding tasks are based on Table 1. Note that the meta-caching methods achieve superior load reductions for bit-rate (Meta-B) and frame-rate (Meta-F) transcoding. For spatial resolution (Meta-S) transcoding, meta-caching is advantageous if the cache size is smaller than 8%. The final-caching method may outperform the meta-caching method when the cache size increases.

Note that if the cache capacity is big enough, the final-caching method always outperforms since no computation is required to deliver cached objects. Another observation is that the final-caching method quickly reduces the average computing load when temporal locality of accessed objects increases. However, in heterogeneous network environments, temporal locality tends to be very low due to the fact that each transcoded variation of the same video object should be considered as a unique transcoded object.

5. CONCLUSIONS & FUTURE WORK

We proposed a meta-transcoding system in which metadata is cached and used as input of subsequent service sessions to reduce the average computing load on the service proxy. We identified metadata for three types of video transcoding tasks. The caching of these types of metadata achieves good tradeoff in storage and computing resources at a transcoding proxy. We showed through simulations that by using meta-caching method, a small amount of cache space enables significant computation savings for bit rate and frame rate transcoding sessions. This is especially useful when memory-based implementation is considered to further reduce the startup latency.

The meta-caching and meta-transcoding concepts can be extended to other types of content services. The key is to identify intermediate steps at which good storage and computing balance can be achieved. For example, consider a language translation service, it may be possible to design an intermediate language representation which occupies less cache space and from which translations to different languages are easily rendered. We can further consider the design of the resource balancer. An adaptive scheme can be designed to decide whether metadata or final results should be cached depending on resource availability and object access characteristics such as temporal locality and frequency. Based on statistic models, thresholds can be derived to facilitate the proxy to render an optimal performance.

6. REFERENCES