Modeling and Optimization of Meta-Caching Assisted Transcoding

Dongyu Liu, Songqing Chen, *Member, IEEE*, and Bo Shen, *Senior Member, IEEE*

***Abstract*—The increase of aggregate Internet bandwidth and the rapid development of 3G wireless networks demand efficient delivery of multimedia objects to all types of wireless devices. To handle requests from wireless devices at runtime, the transcoding-enabled caching proxy has been proposed to save transcoded versions to reduce the intensive computing demanded by online transcoding. Constrained by available CPU and storage, existing transcoding-enabled caching schemes always selectively cache certain transcoded versions, expecting that many future requests can be served from the cache. But such schemes treat the transcoder as a black box, leaving no room for flexible control of joint resource management between CPU and storage. In this paper, we first introduce the idea of meta-caching by looking into a transcoding procedure. Instead of caching certain selected transcoded versions in full, meta-caching identifies intermediate transcoding steps from which certain intermediate results (called *metadata*) can be cached so that a fully transcoded version can be easily produced from the metadata with a small amount of CPU cycles. Achieving big saving in caching space with possibly small sacrifice on CPU load, the proposed meta-caching scheme provides a unique method to balance the utilization of CPU and storage resources at the proxy. We further construct a model to analyze the meta-caching scheme. Based on the analysis, we propose *AMTrac*, *A*daptive *M*eta-caching for *Tra*ns*c*oding, which adaptively applies meta-caching based on the client request patterns and available resources. Experimental results show that AMTrac can significantly improve the system throughput over existing approaches.**

***Index Terms*—Adaptation, CPU intensive computing, meta-caching, transcoding.**

I. Introduction

With the increase of aggregate Internet bandwidth and the rapid development of wireless networks, Internet accesses from portable devices, such as PDAs and cell phones, are also growing rapidly. It is not uncommon that users listen to the digital music or watch a football match through their portable devices. However, the increase of these applications [1], [2] also challenges the existing Internet infrastructure, particularly the existing Internet media delivery systems.

Manuscript received December 20, 2006; revised July 29, 2008. Current version published December 10, 2008. The work was supported in part by the U.S. National Science Foundation under Grants CNS-0509061, CNS-0621631, and CNS-0746649. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. John R. Smith.

D. Liu and S. Chen are with the Department of Computer Science, George Mason University, Fairfax, VA 22030 USA (e-mail: dliu1@cs.gmu.edu; sqchen@cs.gmu.edu).

B. Shen is with the vuclip.com team, vuclip.com, Milpitas, CA 95035 USA (e-mail: bo.shen@ieee.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2008.2007312

Since portable devices generally have different screen sizes, color depths or connection bandwidth from traditional desktop computers, a media object (e.g., a movie) that is good for desktop computers cannot be directly displayed on a PDA. It must be customized appropriately beforehand or at runtime. This type of customization for typical QoS support is often referred to as *content adaptation*.

Two approaches are typically used for providing this type of QoS support in the context of multimedia content delivery. The first approach is called *precoding*. Given an object, this approach either creates multiple provisioned versions or scalably encodes the object with multiple layers or descriptions. All the object versions/layers/descriptions are created before they are ever delivered. For example, many content hosts encode their video clips at different bit rate versions, for example, 28/56 Kbps for dial-up clients and 100-plus Kbps for broadband clients [15]. If considering all possible requirements of client devices (not limited to various network speeds), precoding demands a huge amount of storage for different versions. Scalably-coded content requires less space than multiple individual versions, but it is still not efficient in compression. In addition, content created by this way can only satisfy certain coarse granular QoS requests. It is less flexible when finer granular QoS is required. More importantly, precoding does not scale to the vast variety of media adaptation applications. It may be easy to previse possible bit rate versions that are required for a streaming application, but it would be difficult to precode content for more generic adaptation tasks such as personalization. In the case of overlaying an end user's logo on a video stream, the hosting server is not likely to have the end user's logo available. Thus precoding in this case is impossible.

The second approach, referred here as *transcoding*, offers online real-time adaptation support. Overall, transcoding is not restricted to content customization for QoS support. Being real-time and online, this approach offers more flexibility and scales well with the variety of the applications. However, transcoding is often computing intensive, especially for multimedia content. Research on developing efficient real-time transcoding algorithms has received much attention, especially on video transcoding [13]. From the system perspective, caching is also a viable technique to achieve computing load reduction. The transcoded result for a request can be cached so that future identical requests can be served without transcoding. This kind of transcoding-enabled caching designs has been investigated and the focus has been on efficient utilization of different resources (e.g., CPU, storage, bandwidth) to improve the throughput of the transcoding proxy (see Section II). All of the existing designs treat the transcoding unit - transcoder - as a black box. The cached data is either the input and/or the output

of the transcoder. Specifically, if a transcoded version is fully cached (a *full-caching* scheme), identical future requests can be directly served without additional transcoding. However, to cache each transcoded version may quickly exhaust the cache space. On the other hand, if a transcoded version is not cached (a *no-caching* scheme), identical requests will result in repetitive transcoding, consuming extensive CPU cycles.

In this paper, we propose a meta-caching scheme in which, intermediate transcoding steps are studied and identified so that appropriate intermediate results (called *metadata*) can be cached. With the cached metadata, the fully transcoded object can be easily produced with a small amount of CPU cycles. Interested readers can refer to [18] regarding how to extract metadata for different transcoding processes as well as how to re-produce the final object with the extracted metadata. Since only metadata is cached, the required cache space is greatly reduced. The saved cache space can be used to store metadata for other transcoding sessions so that, in certain conditions, the overall computing load can be reduced. Note that the meta-caching scheme allows the system to achieve a joint control of the CPU and storage resources. It offers a tradeoff point in between what can be achieved by the full-caching and no-caching schemes.

To precisely characterize the meta-caching scheme, we construct an analytical model and investigate the conditional advantages of meta-caching over full- or no-caching. Based on the analysis, we propose a system called AMTrac, which stands for *A*daptive *M*eta-caching for *Tran*s*c*oding. In AMTrac, the meta-caching scheme is adaptively used upon dynamic client accesses and the available resources. Simulation-based experiments show that AMTrac can effectively improve the system throughput over existing schemes.

The rest of the paper is organized as follows. Section II describes some related work. We introduce the generalized meta-caching concept in Section III, and present an analytical model of it in Section IV. An adaptive meta-caching design based on the model is provided in Section V and its performance is evaluated in Section VI. We make concluding remarks in Section VII.

II. Related Work

For scalably-coded content, caching schemes have been investigated in [16]. Layers of precoded content is adaptively cached for optimal server traffic reduction. The relative advantages of caching precoded versions versus layers are evaluated in [8], [11], [14]. In these studies, no online transcoding is needed, the computing load on the proxy is not a concern.

In the online transcoding context, an application level gateway is proposed to consider the transcoding [4]. Middleman also considers transcoding in cooperative proxy servers [3]. Other existing approaches [5], [10], [12], [23] always consider resource utilization of the CPU, storage, and the network by proposing heuristic solutions. The TeC system [19] has proposed strategies to selectively cache original and/or transcoded objects. The adaptive solution proposed in [21] can dynamically select an appropriate metric for changing the management policy. Tu *et al.* [22] proposed an iterative greedy algorithm to minimize utility loss in content distributing systems. The energy-aware strategy has been considered by modeling [20]. Transcoding and scheduling are considered together in a network-attached disk architecture [17].

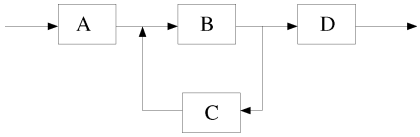


Fig. 1. Example processing flow.

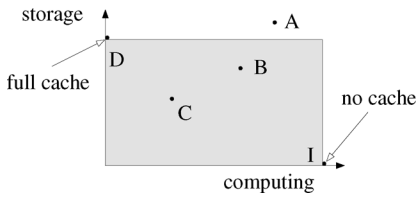


Fig. 2. Mapping into the SC space.

All of these existing designs treat transcoder as a black box. The cached data is either the input and/or the output of the transcoder. Work [18] has investigated the possibility of caching intermediate results from transcoding processes. It has shown for three specific transcoding cases the possibility of reducing the aggregated computing load on the proxy. However, the work is restricted to video transcoding applications and without a dynamic scheme that works for varying client access patterns.

III. Principle of Meta-Caching

To illustrate the principle of meta-caching, we first define a storage versus computing space. Then we discuss some practical applications that can take advantage of this idea.

A. Storage-Computing (SC) Space

Given any media adaptation process, meta-caching is defined as the caching of intermediate results that are created during the course of the adaptation process. As an example, Fig. 1 defines the flow graph of a certain media adaptation process composed of four computing submodules. Caching the intermediate result from each of the four submodules leads to skipping that submodule in the next identical session so that the computing of that submodule is saved. However, some storage space is required to store the intermediate result. In general, caching the output of each submodule maps to one point in a space, called *Storage versus Computing space*. Fig. 2 illustrates such a mapping for this particular process. The vertical axis indicates the amount of storage required to store the intermediate results from any of the submodules, relative to storing the final processed results. The horizontal axis indicates the amount of computing load required to create the final result with the help of the intermediate results from any of the submodules. This computing load is relative to the computing load when the intermediate result is not available, i.e., the computing load of the full adaptation process. Clearly, if the intermediate result of any submodule is directly available (from a cache, for example) instead of computing it from the input, the computing load required to create the final output is reduced.

Now we discuss some of the specific points within the SC space. If nothing from the flow is cached, the point I (no cache)

indicates no storage requirement. But 100% of the CPU is required when a final adapted output is requested. On the other hand, if the final output is cached (full cache), 100% of the storage is required while no computing is needed. Note that point D and point I define a shaded area in this SC space. In general, any point that falls outside the shaded area has no advantage over either D or I. For example, point A is outside the shaded area, which indicates that storing the intermediate result from A would cost more storage than storing the result from D. Obviously, this is not as efficient as simply storing the final result.

Points B and C can introduce certain advantages since both points indicate reduced storage requirement at a cost of certain computing load. In general, the point closer to the origin of the SC space presents better advantage since it indicates less computing and less storage requirement to obtain the final result. In this example, point C is clearly a better choice. In other words, point C indicates that submodule C is more computing intensive yet its intermediate result requires less storage.

Although we illustrate this principle through an arbitrary example, it is clear that the principle is general. Once any media adaptation process is defined, a map in the SC space that reflects the storage and computing tradeoff is uniquely obtainable. We will next discuss some real applications in this context.

B. Applications

The principle outlined above has many practical applications. In particular, video transcoding is the most common type of media adaptation applications.

Fig. 3 illustrates the processing flow of the bit rate reduction of a MPEG video, a case of transcoding. With careful selection of the intermediate points within a transcoding process pipeline, meta-caching can be very useful in reducing the aggregated computing load of an adapting server servicing a ground client requests with different access patterns. In this case, caching of the re-quantization scale factor (M_q) could achieve a good tradeoff between storage and computing resource utilization.

The meta-caching principle is not just restricted to video transcoding applications. To name a few other types of media adaptation applications that can benefit from this principle, we consider the following examples.

- Video to keyframe conversion. Instead of a full length video, a sequence of representative keyframes from the video can be delivered in situations when a client does not have a video player available. This conversion consists of a keyframe analysis process followed by the assembling of the keyframes. Keyframe analysis detects scene changes within a video sequence and identifies frames that are representative of the scene. This can be very computing intensive. However, if the intermediate result (e.g., the frame index of the keyframes) of the keyframe analysis module can be stored, which costs very little storage, the computing load can be significantly reduced for future sessions.
- Personalized logo insertion. With a customized logo inserted into each frame of a video, a client can personalize his/her own content. In this process, the compressed video is first adapted with the logo insertion area to be independently coded. Then a logo is inserted. The adaptation from dependently coded original video to independently coded area can be computing intensive. If the converted independent area can be stored, future sessions can be free of significant computing load. On the other hand, an independently coded area (for example, four significantly smaller corner areas) costs less storage than the full logo inserted video.
- Privacy protection. In this scenario, certain features from content (e.g., certain faces in a video) are automatically blocked to protect privacy. The intermediate results from face detection (e.g., face location on each frame) can be stored so that future sessions can be relieved from the computing intensive face detection process.

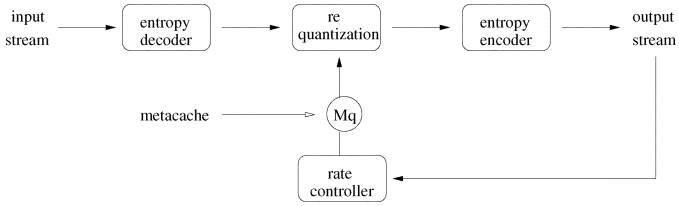


Fig. 3. Bit rate reduction.

For any real-time content processing service, if we can identify from its processing flow a point in the SC space that can benefit from the meta-caching principle, it is possible to improve the overall system performance by balancing the use of storage and computing resources. Since overall system performance depends on aggregated client access patterns, certain points in the SC space can be more advantageous than others, given available computing and storage resources.

IV. Performance Modeling

In this section, we model the performance of the meta-caching scheme by comparing it with full-caching and no-caching schemes. Through the modeling, we aim to precisely characterize the meta-caching scheme under the available storage and CPU resources. For this purpose, in the modeling, we compare the throughput of the meta-caching with other two schemes and show that an appropriate scheme can be selected to fully utilize the available CPU and storage resources. The metric used for comparison is the system throughput. It is represented by the number of concurrent sessions the proxy can serve. Clearly, throughput is constrained by the available cache space and CPU cycles.

In the analysis, we make the following assumptions about client accesses and media objects in the proxy system.

1) The object popularity follows a Zipf-like distribution, and the accessing probability of the k^{th} popular object is denoted as $P_k = (1/k^{\theta})/\sum_{i=1}^{M} 1/i^{\theta}$; we further denote $A = (\sum_{i=1}^{M} 1/i^{\theta})^{-1}$, and thus $P_k = A/k^{\theta}$. θ is the skew factor in the Zipf-like distribution. For media objects access, according to existing media workload characterizations and measurements [6], [7], [9], the value of θ is between 0.47 and 0.73. In our analysis, we assume that popular objects are always cached more favorably than less popular ones.

TABLE I
NOTATIONS USED IN THE ANALYSIS

S	The total cache size
C	The total computing cycle
N	The total number of accesses to objects in the proxy system
N_i	The number of accesses to all versions of object i, $N_i = N * P_i$
M	The total number of multimedia objects in the proxy system
v	The number of versions of each object
α	Storage required to cache metadata (relative to full result)
β	CPU used to produce a fully-transcoded version from cached metadata

2) The inter-request arrival pattern follows a Poisson distribution $p(x,\lambda) = e^{-\lambda}\lambda^x/x!$, $x = 0,1,2,\ldots,\lambda$ is the mean arrival rate.
3) The storage space occupied by each original object is one storage unit. Each object has v different versions for portable devices and each version is accessed by clients uniformly. The storage space needed for caching different versions is different. From the highest quality to the lowest quality, each version occupies space of $1,(v-1)/v,(v-2)/v,\ldots,1/v$. In practice, a system can select an appropriate value of v and the corresponding output version based on the physical conditions.
4) The CPU used for any transcoding that results in a fully cached version is 1 unit. For meta-caching, we assume the CPU consumed to produce the final version is β ($0 < \beta < 1$). The storage space used to cache metadata is a fraction α ($0 < \alpha < 1$) of the fully cached object[1].

The notations used in the following analysis are summarized in Table I.

First, we consider the number of sessions that can be supported by the available storage. We assume that among the total M objects, only k most popular objects can be cached in S storage and all their versions get accessed. Among the N accesses, the total number of sessions to the first k objects is thus $\sum_{i=1}^{k} N_i$.

For full-caching, if we denote v_a as the cache occupied by the v versions of each object, these $\sum_{i=1}^{k} N_i$ sessions use $k * v_a$ cache units. Since different versions of each object are accessed uniformly, the total cache space used is $v_a = 1 + (v-1)/v + (v-2)/v + \cdots + 1/v = (v+1)/2$.

Thus the average storage requirement for each session is

$$s_f = \frac{k \times \frac{v+1}{2}}{\sum_{i=1}^{k} N_i}. \tag{IV.1}$$

For meta-caching, since the average storage requirement is α percentage of a unit, the average storage requirement s_m is

$$s_m = \alpha \times \frac{k \times \frac{v+1}{2}}{\sum_{i=1}^{k} N_i}. \tag{IV.2}$$

[1]If $\alpha = 1$, meta-caching is the same as full-caching, where $\beta = 0$; if $\alpha = 0$, meta-caching is the same as no-caching, where $\beta = 1$. For simplicity, we assume the ratios between metadata and transcoded results for different object versions are the same.

For no-caching, only k units are needed to store original objects in the proxy. Thus these $\sum_{i=1}^{k} N_i$ sessions use k cache units. The average storage requirement s_n for no-caching is

$$s_n = \frac{k}{\sum_{i=1}^{k} N_i}, \tag{IV.3}$$

in which $\sum_{i=1}^{k} N_i \approx N * A * k^{1-\theta}/(1-\theta)$.

Given cache size S, the total number of sessions that can be supported by full-caching, meta-caching and no-caching is denoted as z_{fs}, z_{ms}, and z_{ns}, respectively. We have

$$z_{fs} = \frac{S}{s_f} = \frac{2SNA}{(1-\theta)(v+1)k^{\theta}}, \tag{IV.4}$$

$$z_{ms} = \frac{S}{s_m} = \frac{2SNA}{\alpha(1-\theta)(v+1)k^{\theta}} \tag{IV.5}$$

and

$$z_{ns} = \frac{S}{s_n} = \frac{SNA}{(1-\theta)k^{\theta}}. \tag{IV.6}$$

Having considered the storage usage, we now consider the CPU usage of full-, meta- and no-caching schemes.

For full-caching, if the full results of the transcoding of the k^{th} most popular object are cached, the computing load for obtaining all v versions of the k^{th} object is v. Therefore, on average, the computing load for accesses to object i is v and the average computing load for object i is v/N_i. Since there are k objects in the cache, the average computing load for each session is

$$c_f = \frac{\left(\frac{v}{N_1} + \frac{v}{N_2} + \cdots + \frac{v}{N_k}\right)}{k} \approx \frac{k^{\theta} v}{NA(1+\theta)} \tag{IV.7}$$

where $A = (\sum_{i=1}^{M} 1/i^{\theta})^{-1}$.

Given the available CPU capacity C, the number of sessions that can be supported by the full-caching scheme is denoted as z_{fc}. We have

$$z_{fc} = \frac{C}{c_f} = \frac{CAN(1+\theta)}{vk^{\theta}} \tag{IV.8}$$

Accordingly, we have z_{mc} and z_{nc} for meta- and no-caching, respectively.

$$z_{mc} = \frac{C}{c_m} = \frac{CAN(1+\theta)}{vk^{\theta}(1-\beta) + AN\beta(1+\theta)} \tag{IV.9}$$

and

$$z_{nc} = \frac{C}{c_n} = \frac{CAN(1+\theta)}{k^{\theta} + AN(1+\theta)}. \tag{IV.10}$$

Now we consider both resource constraints jointly. If we use sn_f, sn_m, and sn_n to denote the total session numbers for full caching, meta-caching, and no-caching, we have

$$sn_f = \min(z_{fs}, z_{fc}), \tag{IV.11}$$

$$sn_m = \min(z_{ms}, z_{mc}) \tag{IV.12}$$

and

$$sn_n = \min(z_{ns}, z_{nc}). \tag{IV.13}$$

TABLE II
FIELDS OF DATA STRUCTURE

Field name	Field information
v	Version number: the total number of versions of each object
$ms[1..v]$	Meta_size: an array to record the metadata size of each version
$fs[1..v]$	Full_size: an array to record the full data size of each version
$r[1..v]$	References: an array to record the references to each version
$s[1..v]$	Status: whether the object is currently being replaced (0) or being cached (1)
$u[1..v]$	Utility value: the utility value of the object version
P_s	Storage utilization: current storage utilization (%)
P_c	CPU utilization: current CPU utilization (%)

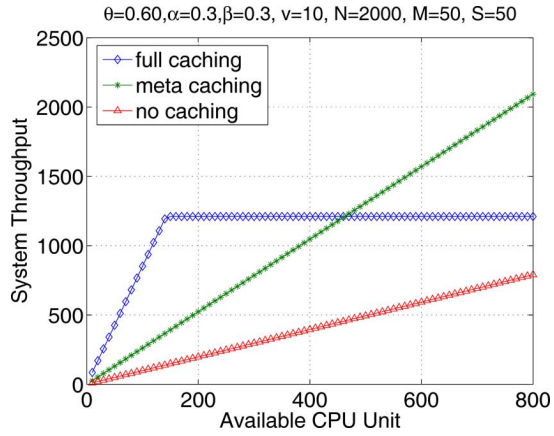


Fig. 4. Comparisons between three methods: available CPU ranges up to 800.

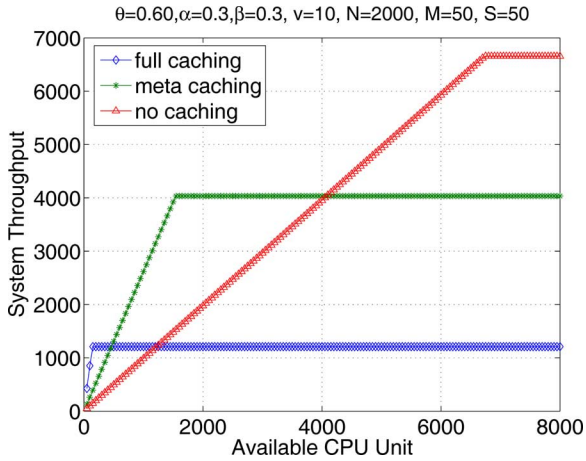


Fig. 5. Comparisons between three methods: available CPU ranges up to 8000.

Based on the above formulas, Figs. 4 and 5 show the comparison results with some typical values for different parameters when the available CPU varies. We omit their detailed comparisons for brevity. In these figures, we set θ to be 0.6, and set α and β as 0.3. We assume that there is a total of 50 objects and a total of 2000 accesses. Each object has ten versions. The total available CPU unit is varying.

In Fig. 4, the available CPU unit ranges up to 800. Fig. 5 extends the available CPU unit to 8000. The figures indicate that under certain conditions, one of the three schemes may outperform the other two.

V. DESIGN OF AMTRAC

We have investigated conditional advantages of meta-caching over other schemes. The result indicates that these three schemes perform differently under different conditions and each may outperform the others. Based on the model-driven analytical result, a system aiming to maximize throughput should thus adaptively use different schemes upon dynamic client access patterns and resource availability. To this end, we present the adaptive meta-caching design for transcoding-enabled proxy, called AMTrac. AMTrac should dynamically select an appropriate scheme with close monitoring of the available storage and CPU resources in the system and client access patterns.

In AMTrac, the proxy provides v different versions of an object to client requests. Each object version is either cached with its metadata only or cached with its fully transcoded result. Thus, in AMTrac, the cache space is logically split into two parts. One is for caching metadata of different object versions, and the other is to cache the fully transcoded object versions. The size of each part changes dynamically.

In AMTrac, the system keeps track of the requested object version, even after an object version is evicted. The data structure for each object is summarized in Table II and we use these data structures to record important runtime information to assist the implementation of a proposed strategy.

In this structure, P_s is calculated as the ratio of the *active disk size* over the *total disk size*. The *active disk size* is the sum of the size of all active (requested) cached objects at present. The *total disk size* is the total available cache size. Based on these, the following three policies work together to implement AMTrac.

A. Progressive Request Admission

Upon a client request for the j^{th} version of an object, there are three cases as follows.

- If the requested object version j is fully cached, the request is directly served and the corresponding data item, $r[j]$, is increased by 1.
- If the requested object version is cached with its metadata, the request is served with the online meta-transcoding (transcoding based on metadata). The transcoded result is sent to the client. The corresponding data item, $r[j]$, is increased by 1. The reactive cache adjustment (see Section V-C) is activated to determine whether this fully transcoded object version should be cached or not.
- If the requested object version does not exist in the cache:
  - If the object version is being accessed for the first time, the proxy performs online transcoding to produce

the object version j. Correspondingly, the metadata is cached. The corresponding data item, $r[j]$, is increased by 1. $ms[j]$ gets updated. The status of this object version $s[j]$ is set to 1. If there is insufficient cache space, the replacement policy (see Section V-B) is activated to make room for its caching.

— If the object version has been accessed and its status is replaced ($s[j] = 0$), the object version is transcoded again and sent to the client. $r[j]$, is increased by 1. The reactive cache adjustment (see Section V-C) is activated to determine whether the corresponding metadata should get cached.

B. Comprehensive Replacement Policy

When there is not enough cache space, replacement is activated. To maximize cache performance, it is important to select the right victim.

In our design, the utility based policy is used to select the right victim. The utility function is designed as follows:

$$U(i_j) = \begin{cases} \frac{r_j}{S_j} \times \frac{\alpha_j}{\alpha_j+\beta_j}, & \text{if } P_c \geq P_s, \\ \frac{r_j}{S_j} \times \frac{\beta_j}{\alpha_j+\beta_j}, & \text{if } P_c < P_s. \end{cases} \quad \text{(V.14)}$$

In (V.14), S_j indicates the occupied cache space of this object version, where $S_j = \max(ms[j], fs[j])$; r_j is the reference number to this object version; α_j is the storage unit (in percentage) used for caching the metadata of version j, while β_j is the CPU unit to transcode to the final version j; P_c represents the current CPU utilization, and P_s indicates the current storage utilization. In this equation, r_j/S_j considers that if an object version is more popular with a unit storage, the object version should have a high utility value and has a better chance to be cached. If $P_c > P_s$, the current system is CPU constrained, so the utilization of the storage is encouraged. If $P_c < P_s$, the system is storage constrained, and the utilization of the CPU is encouraged.

Thus, the utility of an object version considers both the object popularity and the current available resources to find the least valuable object version. Each time the replacement policy is activated, all cached object versions must refresh their utility. Based on the utility function, we design the replacement policy as follows.

When the replacement policy is activated, it compares all the object versions in the cache based on their utility values. The one with the minimum value is selected as the victim and the following procedure loops until sufficient space is found.

- If the selected victim has its fully transcoded data cached in the proxy, the system selects to evict its fully cached data and caches its meta data (it consumes a bit more CPU and it is done when the next time a request is received for this object version). $fs[j]$ is set to 0, while $ms[j]$ is updated accordingly.
- If the selected victim has only metadata cached, the metadata is evicted, $ms[j]$ is set to 0. $r[j]$ is set to 0. The corresponding object version status, $s[j]$, is also set to be replaced.

C. Reactive Cache Adjustment

According to our design, for each object version, there could be three possible cases, namely replaced, with metadata cached, or with fully transcoded result cached. To accommodate the dynamic client accesses and thus to maximize the system performance, we design the reactive cache adjustment policy upon different situations. This adjustment is passive since it is always invoked due to new client requests.

- If a currently replaced object version is accessed, the system starts to evaluate whether the current utility of this object version is increased and is large enough to get cached. The new utility is calculated assuming the object version consumes $ms[j]$ for storage. If the utility is larger than the utility of any cached one, the metadata of this object version gets cached and $ms[j]$ and $u[j]$ are set accordingly. The replacement policy is activated if needed.
- If an object version is cached with metadata, the system starts to evaluate whether or not the fully transcoded data of this object version should be cached. Assuming the fully transcoded object version is cached using space $fs[j]$, its corresponding utility value is compared with the current cached object versions. If the new utility of this object version is higher than that of any cached one, the fully transcoded object version gets cached. Its $fs[j]$ and $u[j]$ are updated accordingly. Correspondingly, its metadata gets evicted and $ms[j]$ is set to 0. Additional space is reclaimed with the assistance of the replacement policy when necessary.

VI. Performance Evaluation

In this section, we first perform experiments on a specific transcoding application to capture the practical values of α and β to set up simulations (Section VI-A). Then we run simulations to study the performance of different strategies for rate reduction. We also evaluate different strategies in a general context when conditions vary (Section VI-B).

A. Experimental Parameter Capturing and Simulation Setup

As explained in Section III, bit rate reduction is a typical transcoding process where meta-caching can be applied to cache M_q. We first focus our experimental study on such an application. To capture the real setup for the storage usage (α) and CPU (β), we implemented a full transcoder and a meta transcoder in C with no special optimizations. 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 on MPEG test sequences with spatial resolution 352×240 coded at 25 fps (frames per second). The original video contains I-, P- and B-pictures and is coded at 512 Kbps.

Considering the bit rate reduction transcoding, caching M_q costs storage at 8250 bytes/s. Note that the scaling factor could be represented by 5 bits in most cases. But when it is cached, 1 byte storage space is occupied. Thus, we use 8 bits for M_q and use different M_q for different macroblocks. We believe this setup gives a reasonable upper bound of the overhead incurred by the meta-transcoding. Please refer to [18] for more details. However, bypassing the rate control submodule enables the

transcoder to get the final result with only 45% of the computing load comparing to a full session. With a target bit rate at 128 Kbps, the selection of this meta-caching point represents (α, β) as (0.5, 0.45). Note that in the rate reduction case, the metadata size does not change with the bit rate reduction. Thus, the corresponding α for version 1, 2, and 3 are 0.167, 0.25, and 0.5. The β values for different versions are the same.

Based on these parameters from a real transcoding application, experiments are conducted when the cache size varies from 4% to 60% of the total unique object size and the total amount of CPU capacity is 100 units. Correspondingly, θ is set to 0.73.

In each workload, there is a total of 1000 objects. Each workload contains 20,000 client requests. The client arrival follows a Poisson distribution with the mean arrival rate as 1 request per 10 s and the maximum arrival interval of 50 min. Clients depart randomly after receiving the service for a duration ranging from 5 min to 40 min. For each object, there are four different versions, including the original best quality object – version 0. Version 3 represents the lowest quality version. The storage for version 1, 2, and 3 is 3/4, 2/4, 1/4 of the original object size (version 0), similar to what is used in our analytical model. Their corresponding encoding rates are 512 Kbps, 384 Kbps, 256 Kbps, and 128 Kbps. We consider four bit rates in the system and thus there are three adaptation parameters: 512 kbps to 384 kbps, 512 kbps to 256 kbps, and 512 kbps to 128 kbps. The total unique original object size amounts to 89.9 GB. The total traffic is 1205.8 GB and the access duration lasts for about 34 h.

B. Experimental Results

We conduct experiments based on rate reduction applications. The three major evaluation metrics used in these experiments are *Locally Completed Session*, *Throughput*, and *Average CPU Load*. *Locally Completed Session* represents overall system performance and is the ratio of the number of accesses that are served locally (on the proxy) over the total number of accesses. *Locally Completed Session* consists of two parts. One is the *Locally Completed Session without Transcoding*, which corresponds to the scenario that the requested version is cached. The other is the *Locally Complete Session with Transcoding*, which indicates the scenario where the transcoding (full or metadata-based) is necessary to serve the client request. *Throughput* represents the system throughput along the time line. It is calculated as the number of sessions that the transcoding proxy can handle per time unit. *Average CPU Load* is defined as the ratio of the sum of current CPU load and the total CPU capacity. It can indicate whether the CPU is saturated due to transcoding load.

1) Performance Overview: Figs. 6–8 show the Locally Completed Session and its two components – with and without transcoding, for four different methods when the cache size increases. In these figures, *No*, *Full*, *Meta* represent the no-caching, full-caching, and meta-caching methods we have discussed. *AMTrac* represents our proposed scheme.

As shown in Fig. 6, when considering the total completed sessions in the transcoding proxy, the performance of the four methods is ordered in *AMTrac*, *Meta*, *Full*, and *No*. Fig. 7 shows that the Locally Completed Session with Transcoding for different methods. The trend is similar to the Locally Complete Session, indicating that the transcoding results for meta-caching and AMTrac are dominant in the totally completed sessions. An interesting variation is shown in Fig. 8 when considering the Local Completed Session without Transcoding. As indicated in the figure, when the cache size increases beyond 16%, *Full* outperforms *Meta* and *AMTrac*. This is due to more cached objects in *Full* when the cache size increases.

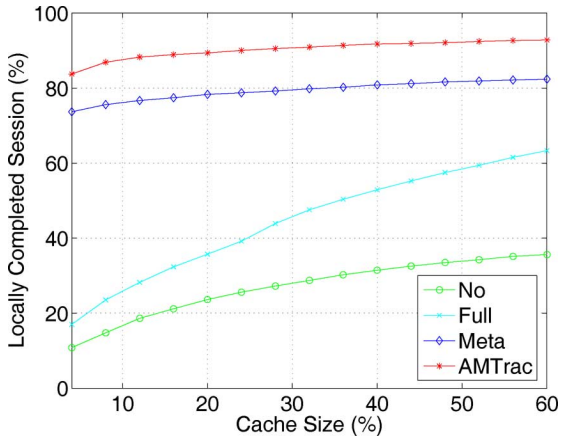


Fig. 6. Total.

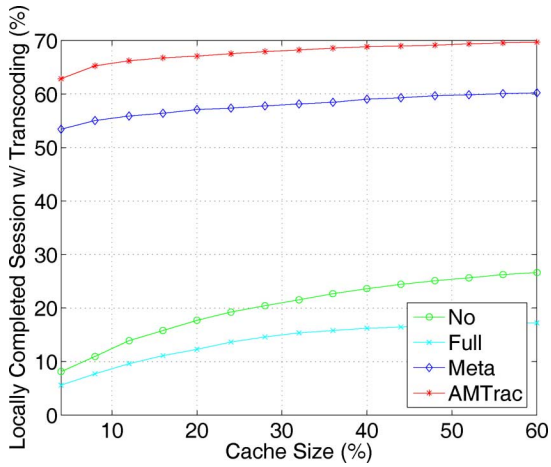


Fig. 7. With transcoding.

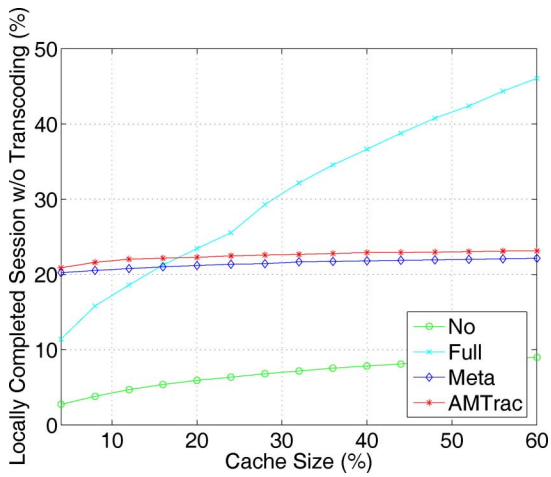


Fig. 8. Without transcoding.

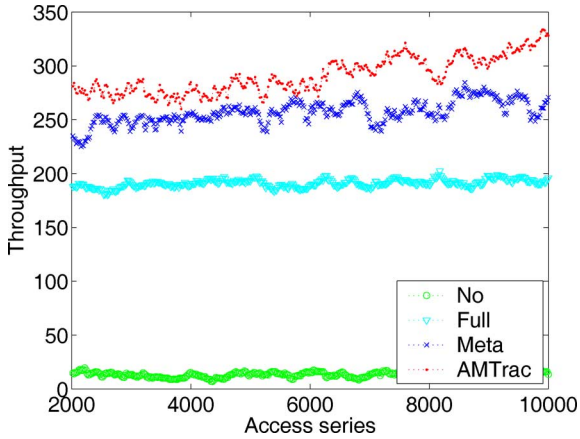


Fig. 9. Throughput.

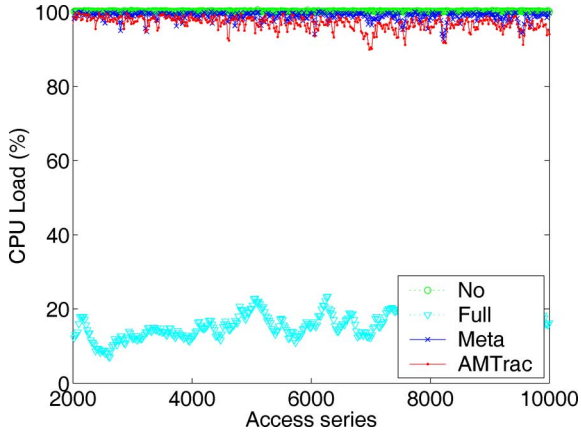


Fig. 10. CPU load.

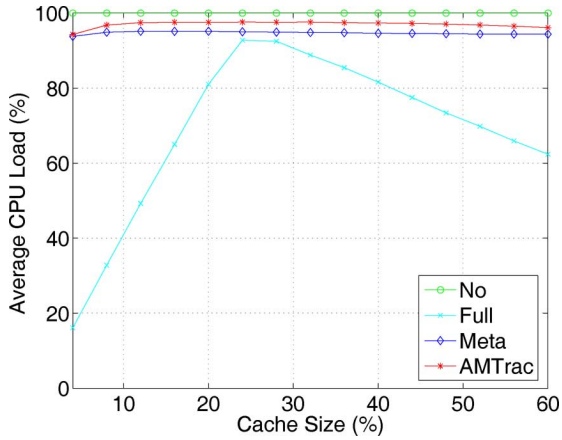


Fig. 11. Average CPU load.

Having examined the overall performance in terms of the total completed sessions in the transcoding proxy, now we examine the proxy's performance at each time unit. Fig. 9 shows the system throughput along the client accesses when the cache size is 4%. Note in the figure, only the results between access 2000 and access 10000 are shown. As indicated in the figure, AMTrac outperforms all other methods. Among all the four methods, no-caching achieves the worst performance as expected and full-caching also has worse performance than that of meta-caching and AMTrac.

Fig. 10 shows the corresponding CPU load along the client accesses (time). Apparently, besides full caching, the other three methods always have a 100% CPU load or close to 100%. To full-caching, since 4% cache space is far from sufficient, its CPU is under utilized. Fig. 11 further shows the average CPU load for the four methods when the available cache size varies from 4% to 60%. As expected, the CPU load of no-caching is not affected when the cache size increases. Since full-caching is the most affected by the available cache space, its average CPU load decreases after the cache size increases beyond 28%. Before that, the limited cache space results in frequent replacement upon new client requests, some of which cannot happen because the selected victim objects are being accessed. Thus, although there are spare CPU cycles, they are not utilized. After the cache size is increased beyond 28%, this situation is relieved. Since full-caching is storage constrained, with larger cache space, more client requests could be served from cache without repetitive transcoding. This is evidenced by the decreasing order of CPU load of full-caching when the cache size is beyond 28%. From another aspect, although meta-caching and AMTrac have a higher CPU load with the increase of cache size, throughput of these two approaches is higher than that of full-caching.

2) Impact of Object Popularity (θ): Existing research reveals that the skew parameter in the Zipf-like distribution is in the range of 0.47 and 0.73 for media objects [6], [7], [9]. In this section, we evaluate the impact of θ on different methods. Three typical values of 0.47, 0.60, and 0.73 are tested while α and β are fixed as 0.5 and 0.45. For brevity, we only show the overall system performance reflected by the Locally Completed Session and its transcoding part.

Figs. 12–14 show the total completed sessions in the proxy for the four strategies when θ varies. In general, these figures indicate θ impacts all methods in terms of the Locally Completed Session, and its impact is more pronounced on full-caching, meta-caching, and AMTrac. This is reasonable since a larger θ indicates more clustered client accesses, and more opportunities for these methods to use cached data to serve incoming requests.

3) Impact of Meta-Caching Parameters ($\alpha + \beta$): The meta-caching parameter β is affected by the CPU clock rate and our previous analysis showed that the system performance is related to the sum of α and β. In this section, we further study how the varying sum of α and β affects the performance of different schemes. In the following experiments α is fixed at 0.5 and θ is fixed at 0.73 when β varies.

Figs. 15, 16 (the same as Fig. 14), and Fig. 17 show the Locally Completed Session for the four strategies. When β varies from 0.3 to 0.6, meta-caching is significantly affected. The larger the value of β, the smaller of the Locally Completed Session. The impact on AMTrac is trivial. The reason is that meta-caching consumes more CPU while AMTrac can adaptively balance the usage of CPU and storage and thus it is less affected. This confirms the effectiveness of our proposed adaptive AMTrac scheme. Since full-caching and no-caching

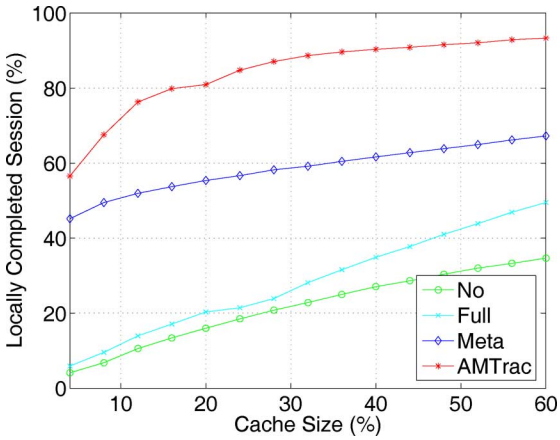


Fig. 12. Total – $\theta = 0.47$.

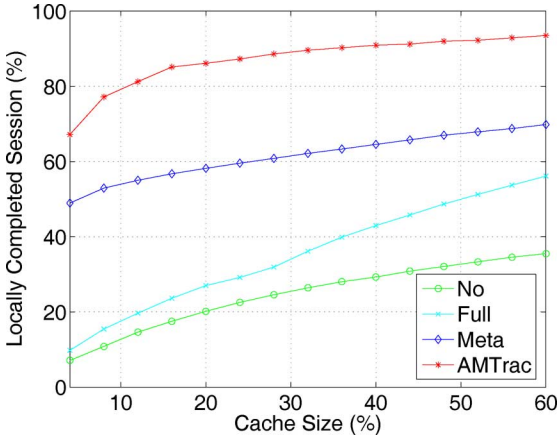


Fig. 13. Total – $\theta = 0.60$.

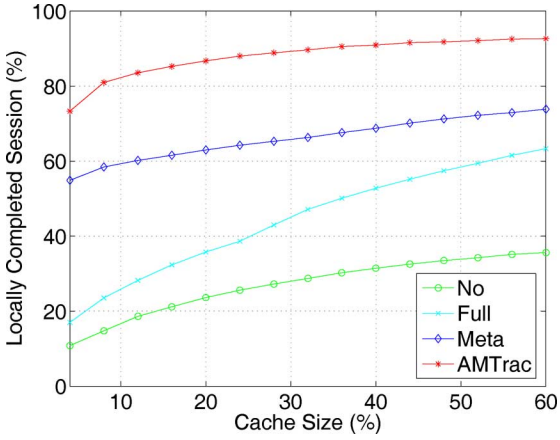


Fig. 14. Total – $\theta = 0.73$.

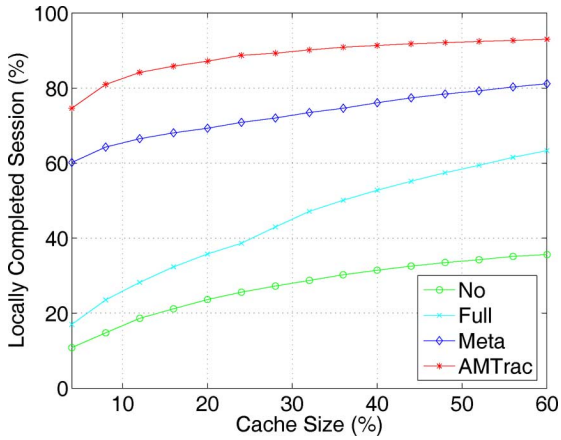


Fig. 15. Total – $\beta = 0.3$.

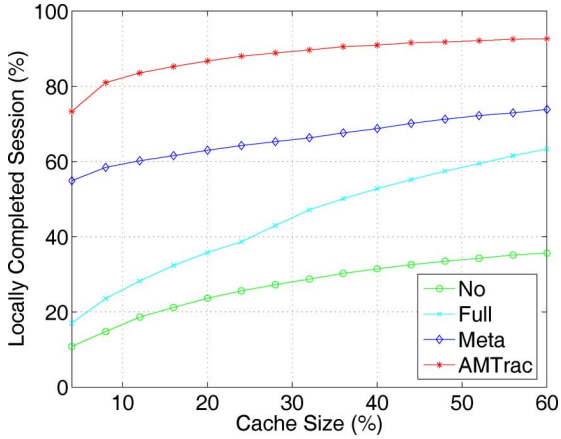


Fig. 16. Total – $\beta = 0.45$.

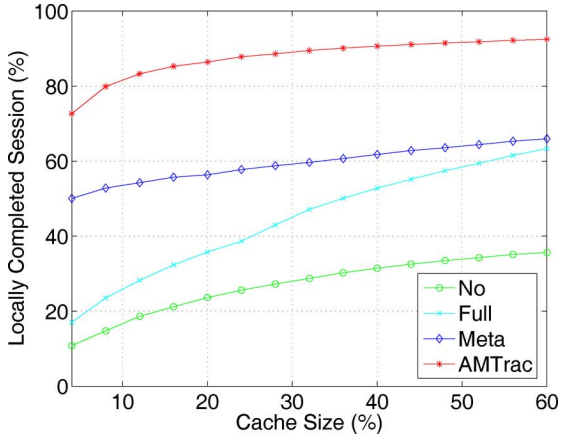


Fig. 17. Total – $\beta = 0.6$.

schemes have nothing to do with β, the performance of these two methods is not affected.

4) Impact of Total CPU: Since meta-caching and AMTrac heavily rely on CPU, we study how the CPU resource affects the performance of different methods. In this section, experiments are run with the total CPU capacity varying from 80 to 120 units. For this set of experiments, α is fixed at 0.5, β is fixed at 0.45 and θ is fixed at 0.73.

Figs. 18, 19 (the same as Fig. 14), and Fig. 20 show the corresponding Locally Completed Sessions for the four strategies when the available CPU varies. When the available CPU units increase, the performance of all methods gets improved. But the improvement of meta-caching and no-caching is more pronounced than the other two. The reason is that meta-caching and no-caching are more dependent on the available CPU resources, while full-caching demands more cache space. Only AMTrac is

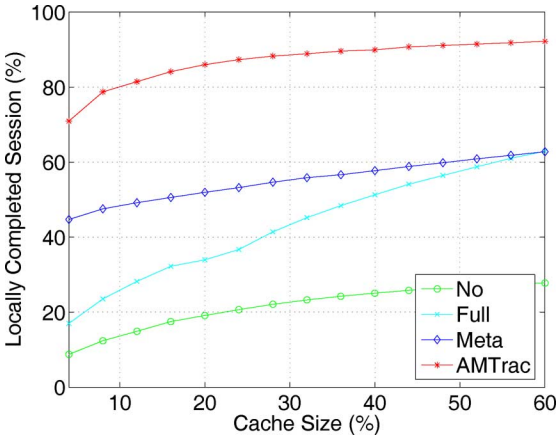


Fig. 18. Total – CPU = 80.

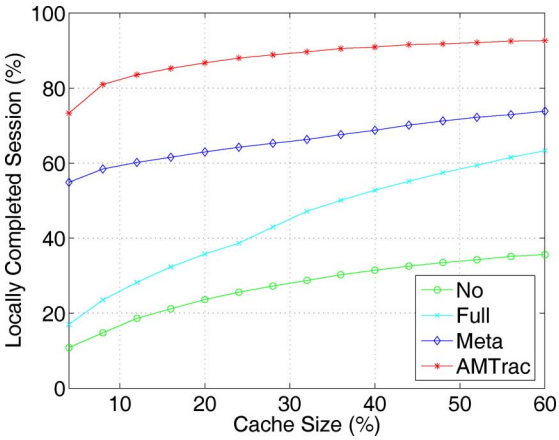


Fig. 19. Total – CPU = 100.

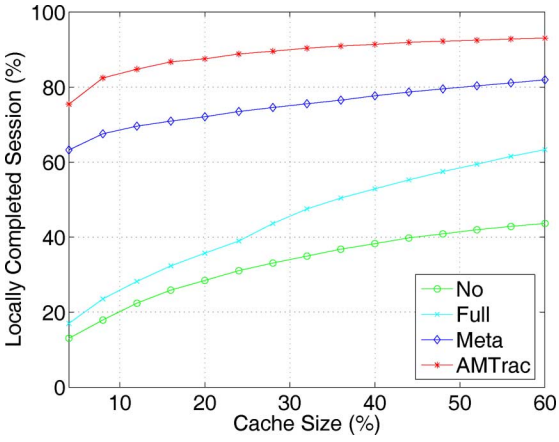


Fig. 20. Total – CPU = 120.

the least affected, particularly when the cache space increases beyond 16%.

These experiments demonstrate that when various conditions, such as available resources and client access patterns, change, our proposed AMTrac can always maintain a balance in using the CPU and the storage to maximize system performance under different conditions.

VII. Conclusion

The Internet has witnessed the rapid increase of Internet media contents and widespread use of portable devices in the past a few years. While a transcoding proxy has been proposed and researched extensively, existing strategies generally aim to reduce the server load and the server traffic. No attention has been paid to the transcoding procedure itself and that leads to less flexibility in addressing the tradeoffs between computing and storage constraints. By proposing to study inside a transcoding process itself, we outline a new approach for caching strategy designs with the main focus on computing load reduction. A meta-caching scheme is proposed that offers a new point of control in the computing and storage space. With model-based analysis on the meta-caching scheme, we propose an adaptive meta-caching system, called AMTrac, which can adaptively use the meta-caching scheme based on client accesses and available resources in the system. Experiments show that it significantly outperforms existing strategies.

Acknowledgment

The authors thank the anonymous referees for providing constructive comments.

References

[1] *Support Nationwide Delivery of Mobile Multimedia*, [Online]. Available: http://www.qualcomm.com/press/releases/2004/041101_mediaflo_700mhz.html

[2] *Your Phone: Mobile Mickey Mouse?*, [Online]. Available: http://www.wired.com/news/wireless/0,1382,64009,00.html

[3] S. Acharya and B. C. Smith, "Middleman: A video caching proxy server," in *Proceedings of ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Chapel Hill, NC, 2000.

[4] E. Amir, S. McCanne, and H. Zhang, "An application level video gateway," in *Proc. ACM Multimedia*, San Francisco, CA, Nov. 1995.

[5] E. A. Brewer, R. H. Katz, Y. Chawathe, S. D. Gribble, T. Hodes, G. Nguyen, M. Stemm, T. Henderson, E. Amir, H. Balakrishnan, A. Fox, V. N. Padmanabhan, and S. Seshan, "A network architecture for heterogeneous mobile computing," *IEEE Pers. Commun.*, vol. 5, no. 5, pp. 8–24, Oct. 1998.

[6] L. Cherkasova and M. Gupta, "Characterizing locality, evolution, and life span of accesses in enterprise media server workloads," in *Proc. ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Miami, FL, May 2002.

[7] M. Chesire, A. Wolman, G. Voelker, and H. Levy, "Measurement and analysis of a streaming media workload," in *Proc. 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, Mar. 2001.

[8] P. D. Cuetos, D. Saparilla, and K. W. Ross, "Adaptive streaming of stored video in a tcp-friendly context: Multiple versions or multiple layers?," in *Proc. Packet Video Workshop*, Kyongju, Korea, Apr. 2001.

[9] L. Guo, S. Chen, Z. Xiao, and X. Zhang, "Disc: Dynamic interleaved segment caching for interactive streaming," in *Proc. 25th Int. Conf. Distributed Computing Systems*, Columbus, OH, Jun. 2005.

[10] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas, "Dynamic adaption in an image transcoding proxy for mobile web browsing," *IEEE Pers. Commun.*, vol. 5, no. 6, pp. 8–17, Dec. 1998.

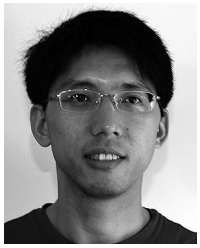
[11] F. Hartanto, J. Kangasharju, M. Reisslein, and K. W. Ross, "Caching video objects: layers vs versions?," in *IEEE Int. Conf. on Multimedia and Expo*, Lausanne, Switzerland, Aug. 2002.

[12] C. K. Hess, D. Raila, R. H. Campbell, and D. Mickunas, "Design and performance of mpeg video streaming to palmtop computers," in *Proc. SPIE/ACM MMCN*, San Jose, CA, Jan. 2000.

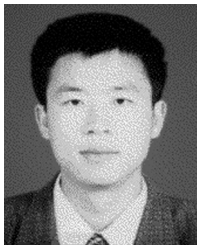
[13] C.-W. Lin, J. Xin, and M.-T. Sun, "Digital video transcoding," *Proc. IEEE*, vol. 93, no. 1, pp. 84–97, Jan. 2005.

[14] T. Kim and M. H. Ammar, "A comparison of layering and stream replication video multicast schemes," in *Proc. ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Port Jefferson, NY, Jun. 2001.

[15] R. Mohan, J. R. Smith, and C. S. Li, "Adapting multimedia internet content for universal access," *IEEE Trans. Multimedia*, vol. 1, no. 1, Mar. 1999.
[16] R. Rejaie and J. Kangasharju, "Mocha: A quality adaptive multimedia proxy cache for internet streaming," in *Proc. ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Port Jefferson, NY, Jun. 2001.
[17] N. Sarhan and C. Das, "Caching and scheduling in nad-based multimedia servers," *IEEE Trans. Parallel Distrib. Syst.*, no. 10, 2004.
[18] B. Shen, "Meta-caching and meta-transcoding for server side service proxy," in *Proc. IEEE Int. Conf. on Multimedia and Expo (ICME03)*, Baltimore, MD, Jul. 2003, vol. I.
[19] B. Shen, S. Lee, and S. Basu, "Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks," *IEEE Trans. Multimedia*, vol. 6, no. 2, pp. 375–386, Apr. 2004.
[20] M. Tamai, T. Sun, K. Yasumoto, N. Shibata, and M. Ito, "Energy-aware video streaming with qos control for portable computing devices," in *Proc. ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Cork, Ireland, Jun. 2004.
[21] X. Tang, F. Zhang, and S. T. Chanson, "Streaming media caching algorithms for transcoding proxies," in *Proc. 31st Int. Conf. Parallel Processing (ICPP)*, Vancouver, BC, Canada, Aug. 2002.
[22] Y. Tu, J. Yan, and S. Prabharkar, "Quality-aware replication of multimedia data," in *Proc. Int. Conf. Database and Expert Systems Applications (DEXA)*, 2005.
[23] T. Warabino, S. Ota, D. Morikawa, M. Ohashi, H. Nakamura, H. Iwashita, and F. Watanabe, "Video transcoding proxy for 3g wireless mobile internet access," *IEEE Commun. Mag.*, vol. 38, no. 10, pp. 66–71, Oct. 2000.



Dongyu Liu received the B.E. and M.E. degrees in computer science from China University of Geosciences in 1997 and 2000, respectively, and the M.S. degree in computational science and informatics from George Mason University (GMU), Fairfax, VA, in 2003. He is currently pursuing the Ph.D. degree in the Computer Science Department at GMU. His research interests include Peer-to-Peer streaming, overlay networks, and content distribution networks.



Songqing Chen (M'03) received the Ph.D. in Computer Science from the College of William and Mary, Williamsburg, VA.

He is an Assistant Professor of Computer Science at George Mason University, Fairfax, VA. His research interests include Internet content delivery systems, Internet measurement and modeling, operating systems and system security, and distributed systems and high performance computing.

Dr. Chen is a recipient of the NSF CAREER Award 2008.

Bo Shen (M'97–SM'04) received the B.S. degre in computer science from Nanjing University of Aeronautics and Astronautics, China and the Ph.D. degree in computer science from Wayne State University, Detroit MI.

He is now a Senior Architect at a mobile video startup, vuclip.com, Milpitas, CA. Before that, he was a Senior Research Scientist with Hewlett-Packard Laboratories. His research interests include multimedia signal processing, multimedia networking and content distribution systems. He has published over 50 papers in prestigious technical journals and conferences. He holds seven U.S. patents with many pending.

Dr. Shen has been on the Editorial Board for IEEE TRANSACTIONS ON MULTIMEDIA from 2006 to 2008. He served as the Lead Guest Editor for IEEE TMM Special Section on Multimedia Applications in Mobile/Wireless Context. He also served on Program Committee for a number of technical conferences including SIGMM.