Impact of Video Encoding Parameters on Dynamic Video Transcoding

Vidyut Samanta, Ricardo Oliveira, Advait Dixit, Parixit Aghera, Petros Zerfos, Songwu Lu

University of California Los Angeles
Computer Science Department
{vids,rveloso,advait,parixit,pzerfos,slu}@cs.ucla.edu

Abstract

Currently there are a wide variety of devices with different screen resolutions, color support, processing power, and network connectivity, capable of receiving streaming video from the Internet. In order to serve different multimedia content to all these devices, middleware proxy servers which transcode content to fit different types of devices are becoming popular. In this paper, we present result of several tests conducted with different combination of video encoding parameters and show their impact on network, CPU and energy resource usage of a transcoding server. For this purpose, we implemented a dynamic video trancoding server, which enables dynamic changes in different trancoding parameters during a video streaming session.

Categories and Subject Descriptors C.2.m [*Computer-Communication Networks*]: Miscellaneous

General Terms Design, Experimentation, Measurement, Performance.

Keywords Dynamic video transcoding, middleware, network bandwidth, processor load, streaming video

1. Introduction

Today there are wide ranges of devices with varying features such as screen size, color depth, and processing power. These devices may have different means by which they connect to the Internet: some may connect through wired 100Mbps connections, some may use wireless technology like CDMA, 1xEV-DO, EDGE, and GPRS, while others may use 802.11b or 802.11g. Hence, the available bandwidth per connection also varies. These conditions make it impractical for content servers to provide different formats of the same content to all different types of devices. Therefore, middleware proxy servers capable of transcoding content for various devices are becoming popular[15, 10, 9, 8].

Due to increase in bandwidth availability, video streaming is becoming a popular method of delivering video content to Internet enabled devices. Popular video streaming services such as Yahoo Launchcast, MSN Video, and MSNBC news web-cast are few such examples. Video streaming can also be used for web-casting lectures or presentations to a number of online attendees at the same time. Client device heterogeneity in terms of screen size, processing capabilities and network connectivity should be handled by a video transcoding server. Moreover, a video transcoding server must be able to address the following scenarios:

1. *Switching between different devices*: Some ubiquitous computing applications allow transferring of a video streaming session from a small PDA device to big screen TV and vice versa. In this case, it is required for transcoding server to change the frame size of the video stream dynamically, which in turn results into change in network and CPU load on the server. Changes in frame size can be disconcerting to the viewer, and so these changes would not be continuous but would only be at isolated discrete moments.

2. *Varying network bandwidth at wireless client*: For devices that are connected in wireless environments, perceived bandwidth changes continuously due to varying characteristics of wireless channels, e.g. fading, interference. This should be handled by changing the transcoding of the content on server dynamically, which results in to increase or decrease in CPU load on the server.

3. *Varying content access patterns*: Access patterns to Internet web sites are highly unpredictable and can have spikes, which may be caused by a particular event like an important sports match or breaking news. For example during a crisis a large number of clients will access streaming clips from a news website. This scenario also results in drastic change in CPU and network load on transcoding server.

Above scenarios indicate that transcoding server is subject to varying network and CPU loads under different circumstances. Since transcoding servers have fixed amount of computing resource and network bandwidth, it should use dynamic transcoding capability to balance the required network and CPU resources with available network and CPU resources without disconnecting the existing client or denying access to new clients. Work in [4, 7] also indicates that energy consumption in server farms is becoming more important as the number of servers increases. This indicates that energy consumption at transcoding server for different transcoding paramters is an interesting relationship to study. One can design an transcoding control algorithm which optimizes the transcoding server resources while serving to maximum number of clients at their acceptable video stream quality. This requires establishing relationship of different transcoding parameters of video stream with network, CPU and energy resources of transcoding servers.

In our work, we perform a systematic evaluation of a transcoding server system by measuring impact of different transcoding parameters such as q-scale, frame size, and color depth on CPU, network and energy resource consumption at the trancoding server and a comparison of transcoding different video content. We also describe our design and implementation of a dynamic video transcoding proxy that we used for this evaluation.

The rest of the paper is organized as follows. Section 2 talks about related works. Section 3 describes the design and implementation of the dynamic video transcoding proxy we have used. Section 4 has experimental results, which show the effect of different transcoding parameters on consumption of transcoding server resources. We conclude in section 5.

2. Related Work

Substantial work has been done in the area of middleware servers. In [5], the authors present a middleware architecture to handle heterogeneous mobile client problems. [3] presents a middleware-oriented solution to the application session handoff problem. Middlware servers have been used to improve scalability in [14]. Seminal work on proxy-based content adaptation is BARWAN project [9]. The argument for adapting data on the fly was made there. A middleware architecture specifically for dynamic video transcoding was presented in [22]. They present a control algorithm, which adapts transcoding to the varying network bandwidth of a 3G-network device. Their algorithm relies on experimental relationship between encoding bit rate and frame rate for a given quantization scale, but lacks data on proxy resource consumption in determining desired transcoding.

We use the cascaded pixel domain transcoding (CPDT) algorithm that was proposed in [16]. The authors of [2] propose a frequency domain transcoding algorithm which significantly reduced computational complexity compared to the CPDT approach. [6] proposes a video transcoding algorithm that integrates error control schemes into the transcoding. They present a two-pass video transcoding scheme that allocates bits between source coding and channel coding.

The effect of transcoding on visual quality has also been studied in literature. A number of studies [11] prove the intuition that sequences with lots of motion require higher frame rate is wrong. Another study [21] shows that large quantization steps should be avoided.

Our work presents experimental measurements of a real transcoding server and can be used in conjunction with these results for designing algorithms for the module that controls transcoding parameters.

3. Design and Implementation

In this section, we describe the design and implementation of an open source based dynamic video transcoding proxy. The design of video transcoding proxy system leverages middleware architecture previously used
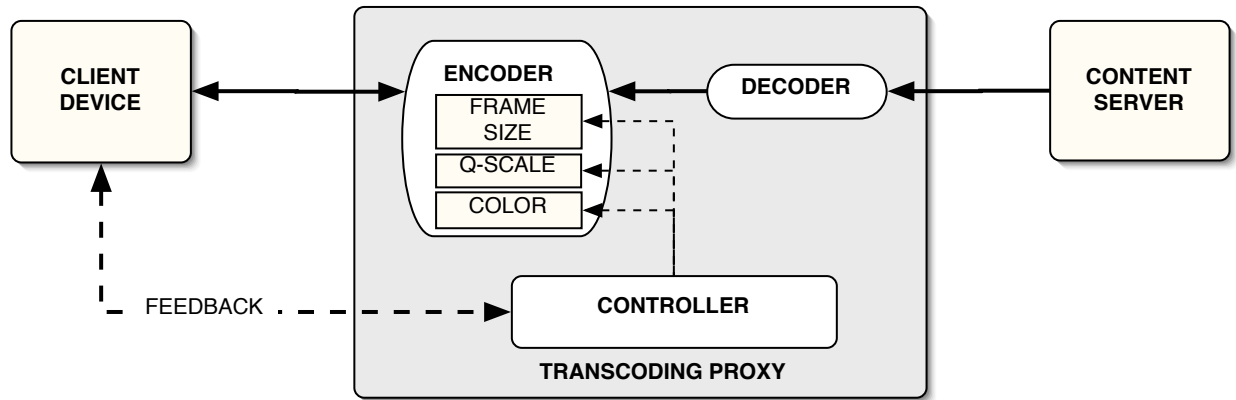
Figure 1. Transcoding architecture

in many transcoding proxy designs. Following is an overview of the transcoding proxy design.

3.1 Design Overview

The design of our video transcoding proxy is illustrated in Figure 1. The system consists of following three components.

1. *The content server* provides high quality video streams to the transcoding proxy. The content server is typically a network of file servers with load balancing mechanisms. Examples of such content servers are CNBC News web-cast server or MSN video server.

2. *The transcoding proxy* is responsible for performing dynamic transcoding of the video content as per the client preferences and different transcoding parameters supplied by a control mechanism. The proxy uses the Cascaded Pixel Domain Transcoder (CPDT) scheme initially proposed in [16], which consists of an encoder connected to a decoder. This particular transcoding scheme is flexible since the frame is fully decoded to raw pixels before re-encoding, which makes dynamic change in transcoding parameters easier. The controller module controls transcoding by changing the frame size, Q-scale (or quantization scale) and color transcoding parameters for encoder. These parameters can be changed using a control algorithm, which takes client device's profile, user's preferences, proxy server resource usage, and network bandwidth available at the client device into consideration.

The controller can have access to client device's profile and user preferences from user database or it can get that information along with the content request. The controller module may get client's available network bandwidth information directly from client device or some by monitoring network conditions. The dotted line in Figure 1 between the client device and the controller is an optional communication link between those modules. Note that design of the control logic as well as control channel communication is a very complex issue and is not addressed in this paper. We are studying the relationship between different transcoding parameters and their impact on transcoding server resource usage, which can be utilized in design of the transcoding control algorithm.

3. *The client* device is used by end user to view the video streaming and is characterized by its screen size, color depth, processing power and network connectivity. Laptops, PDAs and cellphones are examples of some such client devices.

3.2 Transcoding Parameters

Following is the description of the transcoding parameters that can be changed dynamically in our current design.

3.2.1 Frame Size

This is the size of the video frame. It can be initially specified based on the clients screen size. In section 3, we have taken measurements of streaming video with different frame sizes and the results show that smaller frame sizes can take up considerably less network bandwidth. If the client is experiencing network congestion while streaming a video, the parameter on the

transcoding proxy can be dynamically changed so that the size of the frame is reduced and the client receives continuous streaming, but with smaller frame size (Figure 2). If the network conditions improve, frame size can be converted back to size the session started with. One could extend the proxy further to support hand-offs between devices, where the dynamic scaling of the frame size would be very useful for devices with different screen sizes. MPEG standard requires a fresh restart of the video stream in order to change the frame size. In order to circumvent this limitation we keep the original frame size, but insert a black band around the video content as shown in Figure 2.
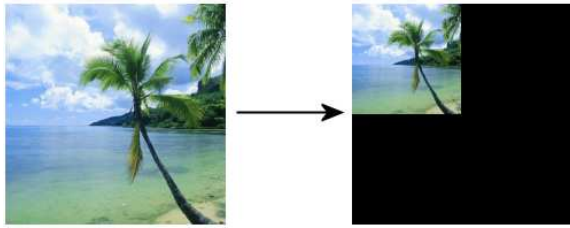
Figure 2. Frame Scaling for width and height by 50%

3.2.2 Color Depth

MPEG standard specifies a single pixel format where each pixel is represented in the YUV scheme, where Y is the luminance, and U and V specifies the chrominance. [12] Because each pixel is represented always with 12 bits (e.g. 4:2:0 planar format), we are only expecting minimal performance improvements by changing the video to gray scale.

3.2.3 Q-scale

The Q-scale is a parameter used to compress a given block of a video frame. Each block is a 8×8 pixel square to which a DCT (Discrete Cosine Transform) is applied (similar to JPEG encoding). The coefficients C_{ij} of each block are then divided by a quantization matrix M_{ij} and a quantization scale q:

$$C'_{ij} = \frac{C_{ij}}{q \cdot M_{ij}} \qquad (1)$$

The encoder quantizes to zero the values of C'_{ij} that are below a certain threshold. Each block is then fed into a RLE module (Run Length Encoder) that does zigzag encoding to achieve better compression. A very high Q-Scale will give a "pixilated" look to a frame (Figure 3) due to the loss of high frequency components

in each frame block. Q-Scale value can be changed from 1 to 31, where 1 indicates highest quality video and 31 indicates lowest quality video.
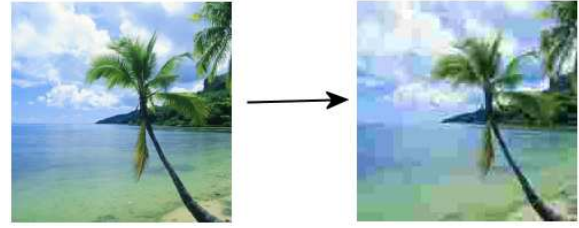
Figure 3. Transcoding operation with a high Q-scale.

3.3 Implementation Details

The video transcoding proxy was implemented using Tinyproxy [20] and FFserver [17]. Tinyproxy is a lightweight http proxy server licensed under GPL. FFserver is a streaming server for both audio and video licensed under LGPL.

FFserver provides two codec libraries ($libavcodec$ and $libavformat$) that can be used to encode and decode MPEG video. We modified tinyproxy and integrated it with the FFmpeg libraries so that it would be able to decode and encode a video stream.

The transcoding process is initiated by the client by sending its device specification, along with the request for a video to the proxy. If transcoding is required, the proxy begins to get the requested video stream and decodes each frame, and then re-encodes it applying the parameters specified by the client during encoding and streams it to the client device. We implemented Q-scaling, frame scaling, and altering color depth while re-encoding the frame at the proxy. The transcoding then works as follows: Tinyproxy has one parent process, which spawns multiple *http server processes* that will serve incoming http requests from clients. We modified tinyproxy so that in addition it spawns an additional child, which serves as a dynamic transcoding request handler. The controller shown in Figure 1 connects to this request handler on the proxy. The controller can dynamically change the video transcoding parameters while streaming.

The algorithm for the transcoder is shown in the flowchart in Figure 4. If the input frame satisfies the transcoding requirements, it is directly sent to the output buffer without being decoded and re-encoded. Otherwise, the frame is decoded. Then, the codec is changed by the transcoder if the transcoding pa-

rameters have changed since the last frame was decoded. The frame is then re-encoded with the current codec and stored in the output buffer, from where it is streamed to the client.
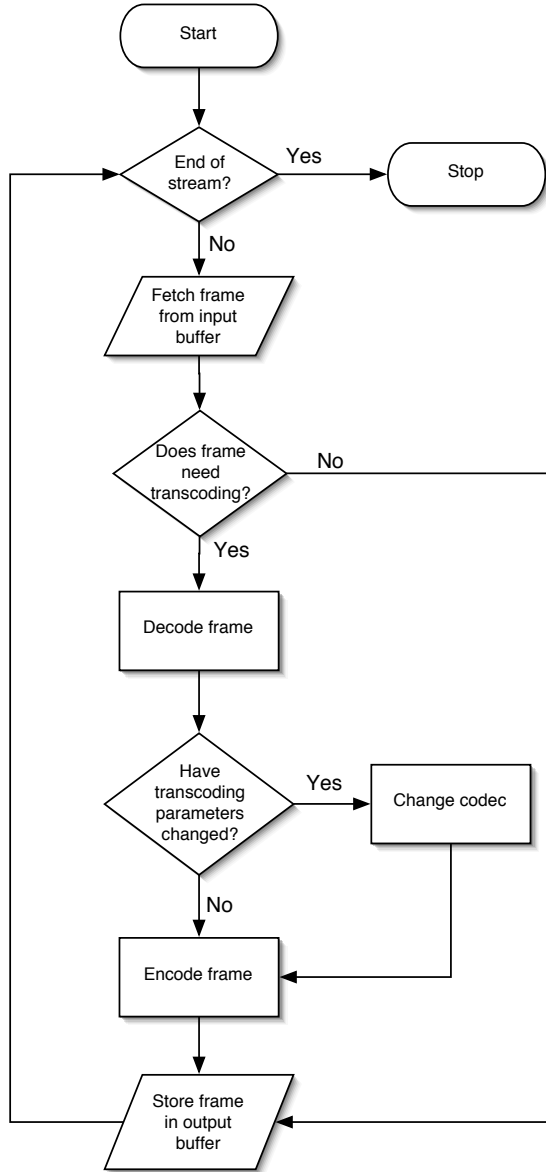
Figure 4. Transcoding Algorithm

4. Measurements

To understand how the trans-coding system reacts to different configuration parameters, we set up a testbed and did the following measurements:

1. CPU usage,
2. Bandwidth usage, and,
3. Energy consumption

4.1   Testbed Setup

For each of the above measurements we used different trans-coding parameters (Q-scale, video frame size and gray/color scale). All the measurements were done at the proxy server. At the client side, we used $MPlayer$ [13] to play the video streams. CPU usage was computed as the CPU time spent by the transcoder during the entire video stream. This CPU time has two components: user time and system time. The user time is the CPU time allocated to the main user-level process and all its children, while the system usage is the CPU time allocated to kernel processes (i.e., disk I/O, network access) during the user process execution [1]. All CPU measurements presented in this paper refer to the total time (user + system).

We used $Tcpdump$ [18] to trace the TCP connections to the server and computed the throughput of these connections using $Tcptrace$ [19].

For energy measurements, we used a laptop with 1.8GHz AMD processor and 512MB of RAM configured as the proxy. Since energy measurements are very sensitive to other processes running in the system at the same time as the transcoder, we turned off all unnecessary processes leaving only the transcoding proxy server and some kernel processes running. The ACPI [1] system of the laptop allowed us to get readings of voltage levels and current drawn from the battery. ACPI can be integrated in the Linux kernel and has a mapping to the $/proc$ file system where current battery status can be accessed. Since ACPI takes measurements directly from the battery, we unplugged the power supply to avoid its constant recharge. We took periodic measurements of voltage level and current intensity from the battery, as the proxy was processing the video stream. We then computed the energy as follows:

$$E(T) = \int_0^T v(t) \cdot i(t) \cdot dt$$

$$\sim \sum_{n=1}^{N} v(n\Delta t) \cdot i(n\Delta t) \cdot \Delta t \qquad (2)$$

Where $v(t)$, $i(t)$ are instantaneous voltage and current drawn from battery, T is the total measurement

---

[1] We used the C function $getrusage()$ to return the values of CPU usage

time, N is the number of samples taken during T and $\Delta t = \frac{T}{N}$.

For the following measurements we used a MPEG-4 encoded video of length 3.5 minutes, with initial frame size $320\text{x}240\,at\,24$ frames a second.

4.2 Q-scale Measurements

For the CPU measurements with different Q-scales we used a 4 minute video clip file in avi format (the video stream was encoded in MPEG-2). We took two measurements for each allowed value of the Q-scale (1-31) and averaged them. The results are shown in Figure 5.
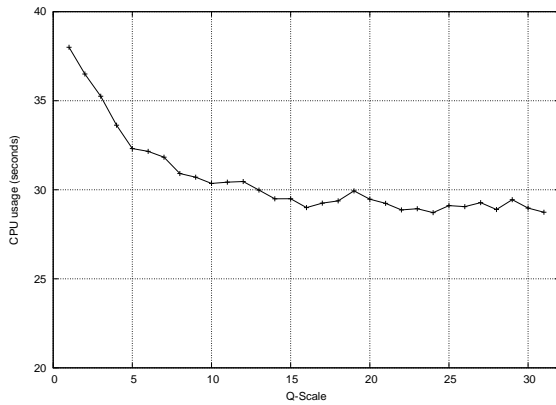

Figure 5. CPU usage for different values of Qscale.

As can be seen from the figure, the total CPU usage has a linear decay for Q-scale<5 and a descendent trend as we go up in the Q-scale. This was expected since MPEG encoding performs a lossy compression over the video image and the amount of compression/loss is greatly affected by the Q-scale parameter. As the value of Q-scale increases, the quantization matrix will contain smaller DCT components, which eventually will be quantized as zero. Since this is followed by a RLE (Run Length Encoding) in each block of the image, a sequence of consecutive zeros will be greatly compressed, thus reducing the time to encode a sequence of video frames (containing B and P frames).

The irregularity of the curves (ups and downs) is due to the fact that we were using a precision of seconds to measure the CPU time, which introduces some level of noise in the measurements (at least one second error margin).

The throughput measurements of Figure 6 were taken using the same video file as above with also two samples per Q-scale:
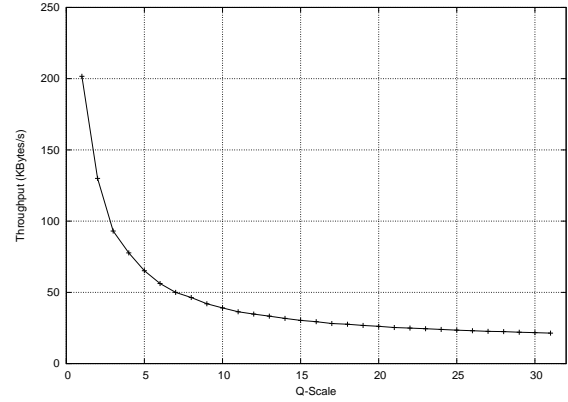

Figure 6. Throughput for different Q-scales.

Figure 6 clearly shows that throughput drops very fast for Q<5 and then converges to a stable value as we increase the Q-scale. This was also expected since as the Q-scale increases, the number of bits necessary to encode the blocks of each video frame is also reduced, thus reducing the amount of data to be transmitted over the network.

4.3 Frame Scale Measurements

MPlayer uses avi decoder functions supplied by the libavcodec library, however the decoder does not support frame size changes during the playback of the video stream. To overcome this limitation we used an image resize function provided by the libavformat library. This function resizes the video image and inserts black strips around the resized image (in the case of a size reduction) but keeps the original height and width of the video frame (Figure 2). Black blocks in a video frame have a high degree of compression ($\sim 100\%$) due to the MPEG image compression scheme (similar to JPEG). In addition, the constant black blocks in the image take advantage of the MPEG motion prediction algorithm to increase the compression level of the video stream and obtain both CPU and bandwidth savings.

Figures 7 and 8 represent the CPU usage (at the proxy) and throughput (at the client) for different frame scales. As an example, after applying a 50% frame scale to a 640x480 frame, the size becomes 320x240, i.e., the scaling applies to both frame width and height. As observed in Figure 7, the CPU usage increases as we approach the original scaling of the frame. Indeed CPU cycles are saved as the original video frame is
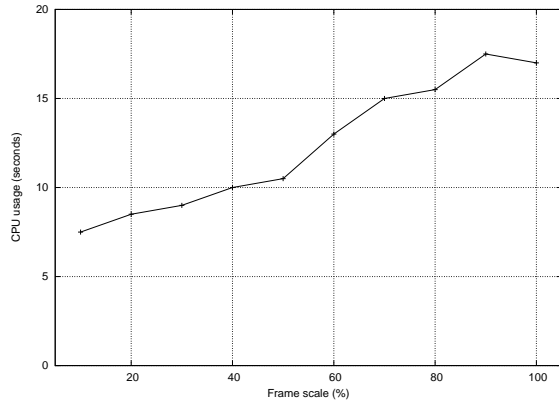
Figure 7. CPU usage for different frame size.

shrunken and the black portion of the frame increases. The encoding will be faster because of:

1. MPEG motion prediction working in the time domain and,
2. MPEG image compression working in the spatial domain.

The same reasoning applies for the throughput curve in Figure 8.

Notice that both CPU usage and throughput curves drop at 100%. This is because with 90% scale, the MPEG encoder still has to encode a tiny black strip on bottom/right sides, however this strip is removed when the frame scaling is set to 100%. The black strip is a discontinuity in spatial domain (high frequencies in DCT) and needs additional bits to be encoded (in comparison to 100% scaling).
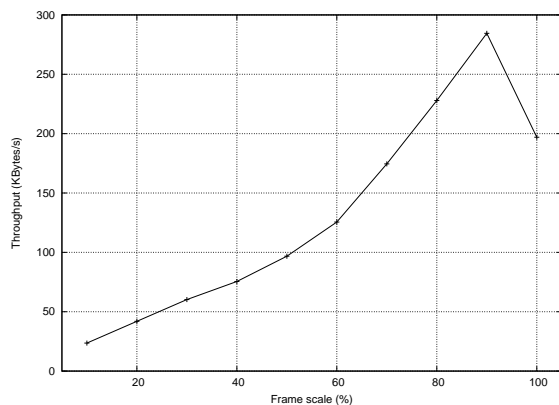

Figure 8. Throughput for different frame scales.

4.4 Gray Scale Measurements

The MPEG standard uses the YUV scheme instead of the traditional RGB scheme to encode each pixel. Y is the luminance component, while U and V are the blue and green chrominance respectively. The most common format is the YUV $4:2:0$ planar format that uses 3 planes (Y, U, V) to encode a given frame. By using gray scale, we are effectively using only the Y plane, since we are discarding the color information. However, MPEG standards require that all three YUV planes exist in the video stream, which means that each pixel is still represented by 12-bit YUV as in the color case. Most of the image information is in the luminance plane (Y plane). Therefore, the throughput and CPU usage are not significantly improved by using gray scale, which were confirmed by our tests.

4.5 Energy Measurements

We did several energy measurements at the proxy, under the conditions described previously in the testbed setup. By changing one transcoding parameter at a time, we could effectively identify the impact of each parameter on the overall energy consumption. Moreover, we were not so much interested in observing the absolute values of the measurements, but to compare energy consumptions of different configurations. These measurements can also extend to other systems like desktops and server farms [7, 4]. Proxy energy consumption is an important parameter in production middleware systems and large data centers, and need to be minimized as much as possible.
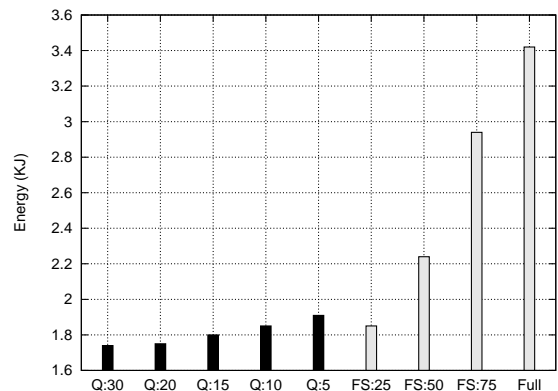
Figure 9. Energy consumption for different transcoder parameters.

Energy was computed according to (2) over a 4-minute video. The frame scale (FS) tests were per-

formed with a Q-scale of 1 and the *Full* test was done without any transcoding at all. Figure 9 shows a behavior similar to that of CPU usage for different transcoding parameters. Indeed, great savings can be achieved by using frame scale reduction or using high Q-scale values.

4.6 Different Video Contents

The impact of transcoding parameters may depend also on the content of the video streams, such as different textures, colors, and movement. We did experiments with four different types of videos that provide a range of different levels of color, textures and motion:

1. *Cartoon*: video clip with flat colors, with fewer textures.
2. *News*: interview between a news caster and a reporter, the background behind the both the news caster and the reporter is static and there is very little movement in each frame.
3. *Sports*: Formula One race, with fast moving cars, there is a lot of movement between each frame.
4. *Music Clip*: video with static parts and parts with lots of movement, and has frames with flat colors as well as heavy textures.
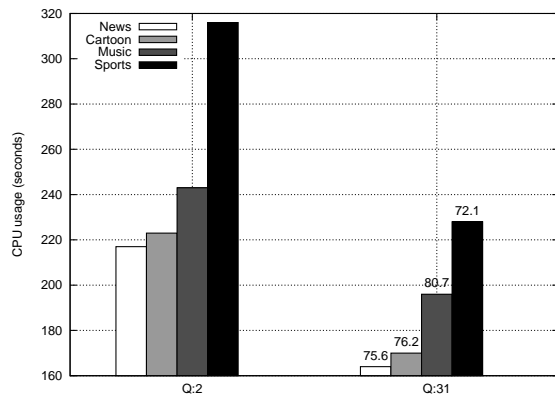
Figure 10. CPU usage for different videos.

All the videos were MPEG-4 encoded and each clip was of length 45 seconds, with initial frame size 640x480 at 30 frames per second. For each video, we compared the performance in terms of CPU and throughput between an encoding with Q-scale=2 and an encoding with Q-scale=31. We expect to observe a decreases in both CPU and throughput usage when going from Q-scale=2 to Q-scale=31, but would this decrease be the same for all videos?
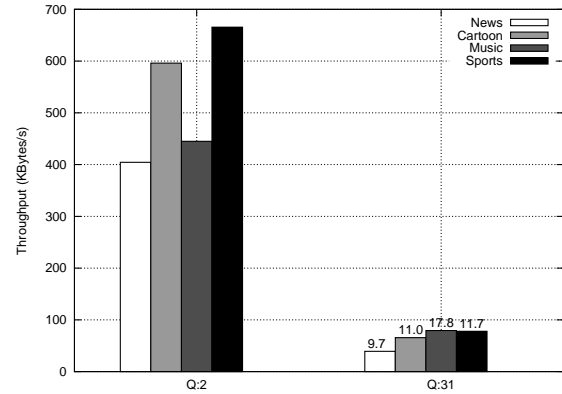
Figure 11. Throughput for different videos.

Figure 10 shows the results for CPU usage: the numbers above the bars represent the percentage of CPU usage of a transcoding with Q-scale=31 compared to the values achieved for Q=2, e.g. a bar with 80% means that trancoding the video with Q-scale=31 takes 80% of the CPU usage of the transcoding with Q=2.

Note that the values are approximately the same for all videos (between 72% and 80%) , indicating that the result of variation of encoding parameters is not impacted by the content of the video.

A similar reasoning can be applied to Figure 11, where we show the throughput measured at the server for different videos.

5. Conclusion

We presented the design and implementation of an open source dynamic video transcoding proxy server based on a three-tier transcoding proxy architecture. The video transcoding proxy provides flexibility for applying different transcoding control algorithms without modifying the other components of the system.

We conducted a systematic evaluation of the system parameters: Q-scale, color depth, and frame size on CPU usage, bandwidth usage, and energy consumption of the transcoding proxy.

Our measurements show that by reducing the image quality down to a Q-Scale of 10, one can dramatically decrease proxy CPU cycles and streaming throughput. Furthermore, video frame size reduction results in a significant decrease of perceived throughput at the client side. Our measurements also show that a 50% scaling on both, width and height of the video frame effectively reduces the throughput to approximately 50%. In addition, energy measurements show that by using a

Q-scale of greater than 5, it is possible to reduce energy consumption at proxy server by more than 50%, in comparison to the no-transcoding scenario. We also study the impact of transcoding of different video content and find that the result of variation of these encoding parameters is not impacted by the content of the video.

We plan to develop an automatic controller module based on our measurement results. The automatic control module would enable us to explore different feedback mechanisms to estimate the client's perceived throughput and experience during the transcoding process and this remains as future work.

References

[1] ACPI at Intel. `http://developer.intel.com/technology/iapc`.

[2] P. A. A. Assuncao and M. Ghanbari. A frequency-domain video transcoder for dynamic bit-rate reduction of mpeg-2 bit streams. In *IEEE Transactions on Circuits Systems Video Technology, vol. 8, no. 8, pp. 953-967*, December 1998.

[3] R. Bagrodia, S. Bhattacharyya, F. Cheng, S. Gerding, G. Glazer, R. Guy, Z. Ji, J. Lin, T. Phan, E. Skow, M. Varshney, and G. Zorpas. imash: Interactive mobile application session hando, 2003.

[4] G. Banga, P. Druschel, and J. Mogul. Resource containers: A new facility for resource management in server systems. In *Proceedings of the Third Symposium on Operating System Design and Implementation OSDI'99*, February 1999.

[5] A. Bond, M. Gallagher, and J. Indulska. An information model for nomadic environments. In *DEXA Workshop*, pages 400–405, 1998.

[6] J. Cai and C. W. Chen. A high-performance and low-complexity video transcoding scheme for video streaming over wireless links. In *Proceeding of Wireless Communications and Networking Conference 2002*, March 2002.

[7] J. Chase, D. Anderson, P. Thakur, and A. Vahdat. Managing energy and server resources in hosting centers. In *Proceedings of the 18th Symposium on Operating Systems Principles SOSP'01*, October 2001.

[8] A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 160–170, Cambridge, MA, October 1996.

[9] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *Proceedings of the 16th Symposium on Operating Systems Principles (SOSP-97)*, volume 31,5 of *Operating Systems Review*, pages 78–91, New York, Oct.5–8 1997. ACM Press.

[10] R. Han, P. Bhagwat, R. LaMaire, T. Mummert, V. Perret, and J. Rubas. Dynamic adaptation in an image transcoding proxy for mobile www browsing. *IEEE Personal Communication*, 5(6):8–17, December 1998.

[11] J. D. McCarthy, M. A. Sasse, and D. Miras. Sharp or smooth?: comparing the effects of quantization vs. frame rate for streamed video. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 535–542, New York, NY, USA, 2004. ACM Press.

[12] MPEG Home Page. `http://www.chiariglione.org/mpeg/`.

[13] MPlayer. `http://www.mplayerhq.hu`.

[14] T. Phan, R. G. Guy, and R. Bagrodia. A scalable, distributed middleware service architecture to support mobile internet applications. In *Wireless Mobile Internet*, pages 27–33, 2001.

[15] A. Singh, A. Trivedi, K. Ramamritham, and P. J. Shenoy. Ptc: Proxies that transcode and cache in heterogeneous web client environments. *World Wide Web*, 7(1):7–28, 2004.

[16] H. Sun, W. Kwok, and J. Zpedski. Architectures for mpeg compressed bitstream scaling. In *IEEE Trans. Circuits and Systems for Video Technology*, April 1996.

[17] FFmpeg. `http://ffmpeg.sourceforge.net`.

[18] tcpdump / libpcap. `http://www.tcpdump.org/`.

[19] tcptrace. `http://www.tcptrace.org/`.

[20] Tinyproxy. `http://tinyproxy.sourceforge.net/`.

[21] D. Wang, F. Speranza, A. Vincent, T. Martin, and P. Blanchfield. Toward optimal rate control: a study of the impact of spatial resolution, frame rate, and quantization on subjective video quality and bit rate. In *VCIP*, pages 198–209, 2003.

[22] A. Warabino, S. Ota, D. Morikawa, M. Ohashi, H. Nakamura, H. Iwashita, and F. Watanabe. Video transcoding proxy for 3gwireless mobile internet access. *Communications Magazine, IEEE*, pages 66 – 71, Oct 2000.