Complexity-Aware Live Streaming System

Meng-Ting Lu, Chang-Kuan Lin, Jason Yao and Homer Chen
Graduate Institute of Communications Engineering
National Taiwan University
Taipei, Taiwan 106, R.O.C.
b88901124@ntu.edu.tw, b0901093@ee.ntu.edu.tw, jasonyao@cc.ee.ntu.edu.tw and homer@cc.ee.ntu.edu.tw

Abstract—**The number of client requests that a streaming server can handle is limited by both its computational resources and available bandwidth. While bandwidth capacity is critical for most streaming applications, computational resources for a server encoding live videos often become a critical factor as well. In order to serve more client requests or provide higher quality for high priority clients, it is desirable to allocate and adjust the computational resources on a per channel basis. In this paper, we proposed a complexity-aware live video streaming server system that manages the computational resources dynamically. In the proposed system, input videos are encoded with different quality levels based on their priorities and available computational resources. The computational resources for each encoder are adaptively allocated to match the time constraints. The seven quality levels defined in the XviD MPEG-4 encoder are used in our experiments. The results show that the new design is able to maximize the resource utilization by maintaining the highest priority channel's quality while providing the other channels with best-effort quality.**

Keywords-component; MPEG-4; RTP/RTCP; video streaming; complexity-aware streaming

I. INTRODUCTION

Because of the prevalence of Internet, streaming video applications have become more and more popular these days. These streaming applications include video on demand (VoD), video conference, IP-based TV, etc. Most of these systems use multimedia standards like MPEG-4 [1] video codec, and protocols like RTP/RTCP [2], RTSP [3], and SDP [4]. The basic form of streaming MPEG-4 content is also discussed in RFC 3016 [5]. Many leading commercial products, such as RealPlayer, QuickTime and Windows Media all conform to these standards. For video on demand applications, the number of client requests that a streaming server can support is usually limited by the bandwidth because the complexity of RFC 3016 is quite low. However, for live streaming, the complexity of the MPEG-4 encoder becomes a critical factor because the number of client requests the server can handle is determined by the computational resources available on the server instead of the bandwidth. Therefore, an effective mechanism for controlling the computational resources of each video channel is needed.

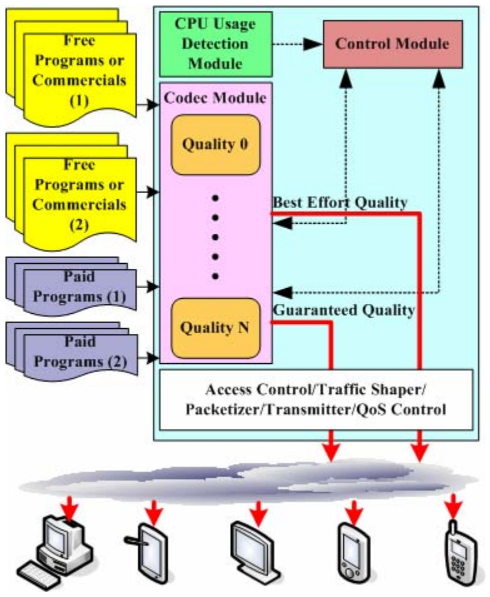
Tai et al. [6] proposed a computation-aware scheme for software-based block motion estimation that allows the searching process to stop once a specified amount of computation has been performed. This scheme is designed for one encoder which has been allocated some computational resources, and the encoder tries to allocate them to each block in a computation-distortion-optimized manner. The scenario considered in this paper is quite different. Here, we have a streaming server with multiple live video channels and encoders, and the resources of the streaming server are limited. The task is to perform a trade-off between video quality and computational complexity for each encoder. We propose a complexity-aware live video streaming system that emphasizes efficient utilization of available computational resources to provide the best quality of service. In our system, input videos are encoded with different quality levels based on their priorities and the computational resources on hand. A novel control module is devised to calculate the time constraint and allocate the computational resources to each encoder accordingly. Under this scheme, we can maintain the highest priority channel's quality while providing the other channels with best-effort quality.

The rest of this paper is organized as follows. Section 2 presents the usage scenario and architecture of our complexity-aware streaming system, while Section 3 presents the details of our control module. Section 4 describes the quality levels and codec used in our system. Simulation results and discussion are presented in Section 5. Finally, Section 6 concludes this paper.

II. USAGE SCENARIO AND SYSTEM ARCHITECTURE

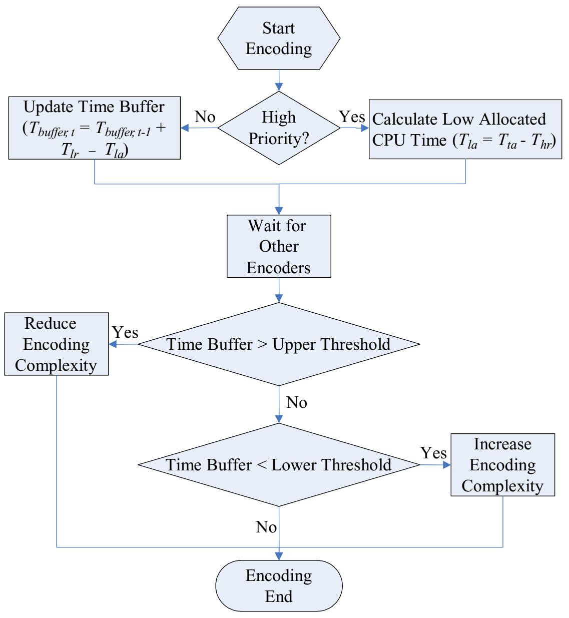
This section describes the usage scenario and system architecture of our system. The system architecture is shown in Fig. 1. Assume a content provider company provides two kinds of live content, paid and free programs or commercials. It is reasonable for a customer to expect better quality when viewing paid programs. Therefore, the quality of free programs or commercials should be adjusted when a streaming server's load reaches a threshold. The control module in our architecture keeps monitoring the CPU processing time of each encoder. When the CPU processing time reaches a predetermined threshold, it signals the codec module of the free programs and commercials to operate with a less complex encoding scheme so that the computational resources can be allocated to serve more customers or to guarantee the quality of the paid programs. With this architecture, paid programs can be served with guaranteed quality while free programs or commercials are served with best-effort quality.

This work was supported in part by the National Science Council of Taiwan under contract NSC 93-2213-E-002-137, NSC 93-2219-E-002-031 and NSC 93-2752-E-002-006-PAE.



CPU Usage Detection Module
Control Module
Free Programs or Commercials (1)
Free Programs or Commercials (2)
Paid Programs (1)
Paid Programs (2)
Codec Module
Quality 0
Quality N
Best Effort Quality
Guaranteed Quality
Access Control/Traffic Shaper/ Packetizer/Transmitter/QoS Control

Fig.1. Complexity-aware streaming server architecture.



Start Encoding
High Priority?
No
Yes
Update Time Buffer ($T_{buffer,t} = T_{buffer,t-1} + T_{lr} - T_{la}$)
Calculate Low Allocated CPU Time ($T_{la} = T_{ta} - T_{hr}$)
Wait for Other Encoders
Time Buffer > Upper Threshold
Yes
Reduce Encoding Complexity
No
Time Buffer < Lower Threshold
Yes
Increase Encoding Complexity
No
Encoding End

Fig. 2. Control module flowchart.

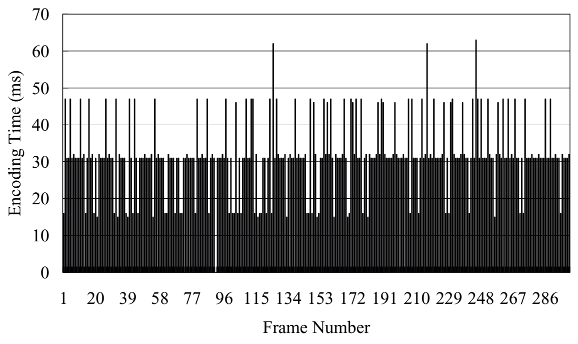


Fig. 3. Encoding time of Stefan at quality level 6.

III. Flow of Complexity Control

This section describes the details of our control module. In our system, the control module is responsible for allocating computational resources. The flowchart of control process is shown in Fig. 2 and explained as follows. First, we decide the desired frame rate for all encoders. Based on the frame rate, we compute the total encoding time T_{ta} for all encoders by taking the inverse of the frame rate. All encoders have to finish encoding one frame within the time interval T_{ta}. Then, we can calculate the available CPU time for all low priority encoders T_{la} based on the real CPU processing time of high priority encoders T_{hr}. After getting T_{la}, we can decide whether to change low priority encoders' quality levels by observing if their encoding time exceeds the allocated time. However, as shown in Fig. 3, the encoding time can vary widely. So, if we change the encoding quality levels every time the encoding time exceeds the allocated time, there will be too many unnecessary quality level changes. Therefore, a time buffer is implemented to smooth the process of quality level adjustment at the encoder. Every time an encoder finishes encoding one frame, it waits until the other encoders finish their jobs. Then the time buffer is updated by adding the difference between the low priority encoders' allocated CPU time and the sum of all their real CPU processing time. When the time buffer exceeds the upper threshold, our system picks a low priority encoder with the highest average PSNR to reduce its encoding complexity. When the time buffer is lower than the lower threshold, our system picks a low priority encoder with the lowest average PSNR to increase its encoding complexity. This process continues until all encoders do not have any more frames to encode.

IV. Quality Levels and XviD Encoder

The section describes the quality levels and MPEG-4 encoder we used for our experiments. We use XviD 1.0.3 MPEG-4 encoder in all our experiments. The XviD implementation of MPEG-4 is an open-source project covered by GPL license. It is written mostly in C, although some of its functions are also written in assembly language. There are seven quality levels in XviD's example encoding program. We take these quality levels as our encoding complexity adjustment basis.

In quality level 0, the encoder utilizes PMVFAST with basic diamond search algorithm. In quality level 1, the encoder utilizes PMVFAST with advanced diamond search algorithm. In quality level 2, the encoder adds half-pel refinement. In quality level 3, the encoder adds the 8x8 mode. In quality level 4, the encoder adds SAD for chrominance. In quality level 5, trellis-based R-D optimal quantization is added. In quality level 6, the encoder adds extra search points and high quality ac prediction. In the following figures, the rate-distortion curves of several test sequences at different quality levels are shown. We can see from these figures that the PSNR values and CPU processing times increase with quality levels at the same bit-rate. All these rate-distortion curves are obtained by performing a series of experiments with only I and P frames and with I frame interval being thirty frames.

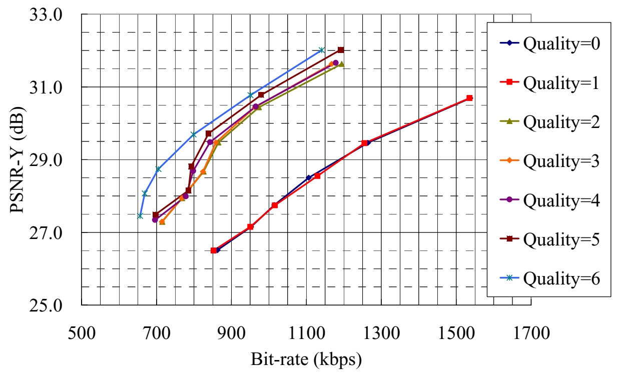


Fig. 4. Rate-distortion curve of Stefan.

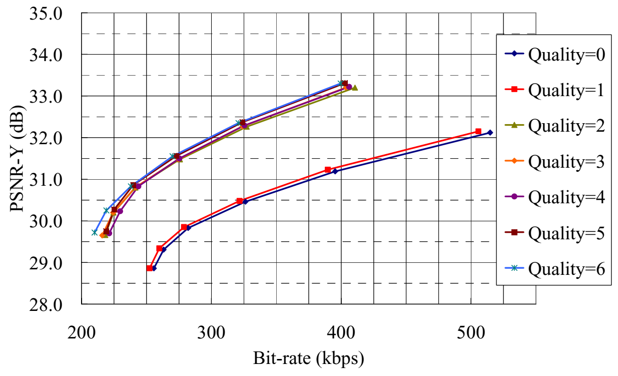


Fig. 5. Rate-distortion curve of Foreman.

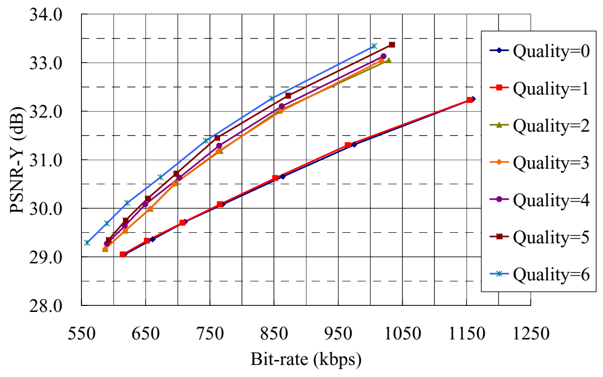


Fig. 6. Rate-distortion curve of Football.

V. SIMULATION RESULT

In this section, the simulation results are presented. We did all these experiments on a computer with Pentium (R) 4 CPU 2.4GHz and 768MB RAM. The test sequences used in our experiments are at CIF resolution. All sequences were encoded with only I and P frames with I frame interval being 30 frames and target bit-rate 1.5Mbps. We set the upper threshold to 80, and the lower threshold to -80 based on our experimental tests. Due to paper length restrictions, these results are not shown here. In the following experiments, one

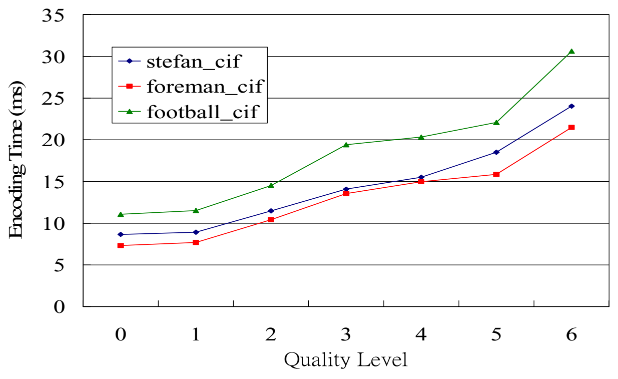


Fig. 7. Average encoding time of each frame.

high priority encoder and two low priority encoders were used, and all encoders were initially defined at quality level 6.

In the first experiment, we defined eight different target frame rates and allocated the computational resources according to the control scheme described above. The results are shown in table 1. The average encoding time in this table means the average time needed for all encoders to finish encoding one frame. Here we also change the target frame rate to target encoding time for each frame. As can be seen from this table, the results of our experiments are quite accurate. In the second experiment, we define the target frame rate to be 20 frames per second. Fig. 8 shows PSNR values of all encoded sequences, and Fig. 9 shows the accumulated CPU processing time of each encoder. We can see from these figures that at first, all encoders initially have the same PSNR and CPU processing time. But when the time buffer exceeds the threshold, the encoding quality levels of low priority encoders are adjusted. Finally, the CPU processing time and PSNR values are much lower than the high priority one. We also show in Figs 10-11 the 230th frame of the Football sequence from a high priority encoder and the 230th frame from a low priority encoder. We can see from the pictures that the high priority encoder's video quality is maintained and better than the low priority video quality.

Table 1. Average encoding time.

Target Encoding Time	Target Frame Rate	Average Encoding Time	Average Frame Rate
50 ms	20.00 fps	52.29 ms	19.12 fps
55 ms	18.18 fps	55.71 ms	17.95 fps
60 ms	16.67 fps	60.29 ms	16.59 fps
65 ms	15.38 fps	66.29 ms	15.09 fps
70 ms	14.29 fps	70.31 ms	14.22 fps
75 ms	13.33 fps	75.55 ms	13.24 fps
80 ms	12.50 fps	80.22 ms	12.47 fps
85 ms	11.76 fps	84.55 ms	11.83 fps

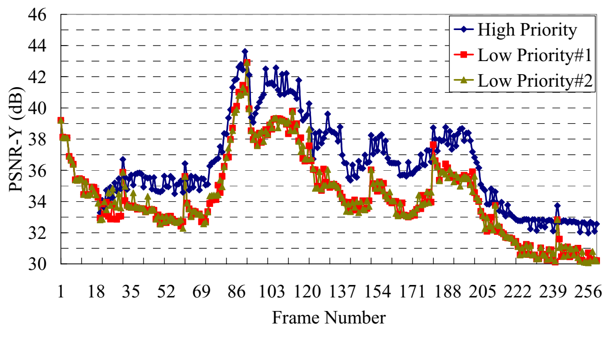


Fig. 8. PSNR values of all frames.

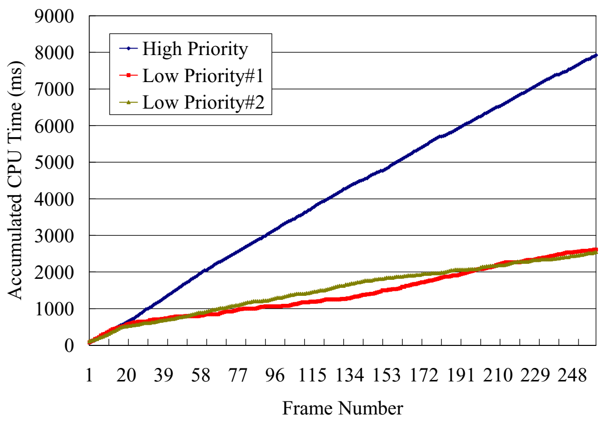


Fig. 9. Accumulated CPU processing time of all encoders.

VI. Conclusion

In this paper, we have presented a complexity-aware live streaming system for better utilization of computational resources for differentiated services. In this system, the complexity of each encoder is adjusted according to the available computational resources. The final simulation results prove that our system is quite accurate in allocating resources under the given time constraints. This system will be very useful for IP-based TV and on-line sports game relay applications. Our future work will be extending our current system to adjust encoder's complexity at the block-level and to support H.264 codec and transcoding applications.

Acknowledgment

This work was supported in part by the National Science Council of Taiwan under contract NSC 93-2213-E-002-137, NSC 93-2219-E-002-031 and NSC 93-2752-E-002-006-PAE.

References

[1] Coding of Audio-Visual Objects: Visual, ISO/IEC Standard 14496-2, 1998.

[2] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," *RFC 3550*, IETF Network Working Group, July 2003.

[3] H. Schulzrine, A. Rao, and R.Lanphier, "Real Time Streaming Protocol (RTSP)," *RFC 2326*, IETF Network Working Group, April 1998.

[4] M. Handley and V. Jacobson, "SDP: Session Description Protocol," *RFC 2327*, IETF Network Working Group, April 1998.

[5] Y. Kikuchi, T. Nomura, S. Fukunaga, Y. Matsui, and H. Kimata, "RTP Payload Format for MPEG-4 Audio/Visual Streams," *RFC 3016*, IETF Network Working Group, November 2000.

[6] P.-L. Tai, S.-Y Huang, C.-T. Liu, and J.-S. Wang, "Computation-aware scheme for software-based block motion estimation," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 13, no. 9, pp. 901-913, September 2003.

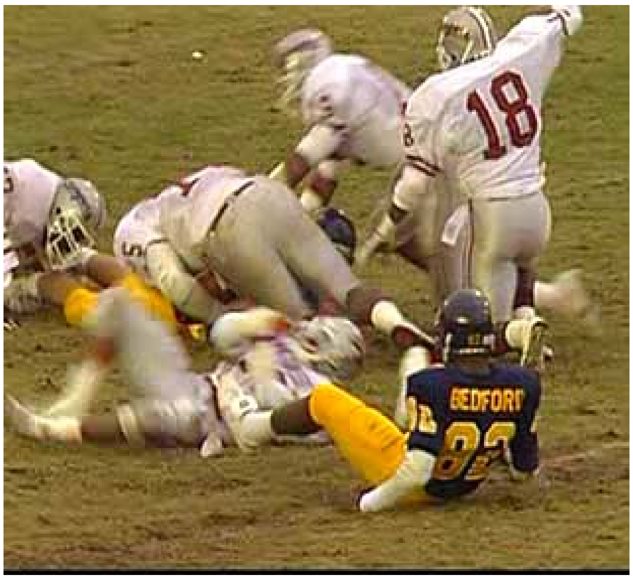


Fig. 10. The 230th frame of high priority encoder.

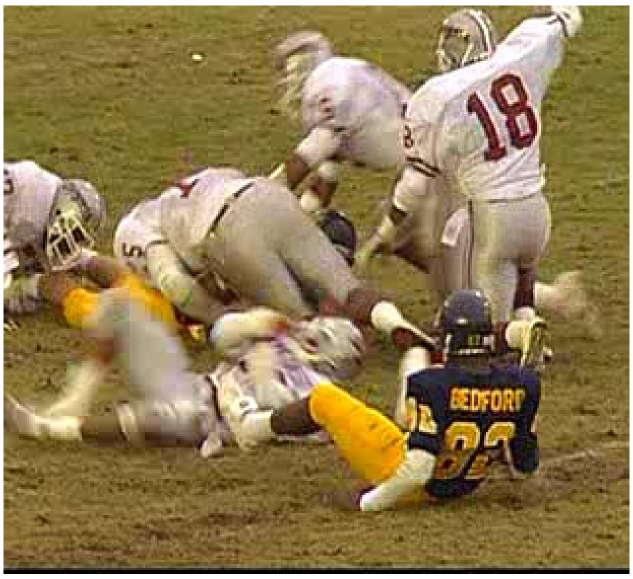


Fig. 11. The 230th frame of the first low priority encoder.