

Adaptive Real-time Monitoring Mechanism for Replicated Distributed Video Player Systems

Chris C.H. Ngan, Kam-Yiu Lam and Edward Chan

Department of Computer Science

City University of Hong Kong

83 Tat Chee Avenue, Kowloon

HONG KONG

Email: cskyalm@cityu.edu.hk

Abstract

One of the main problems to ensure the quality of services (QoS) in video playback in a replicated distributed video player system is how to adapt to the changing network traffic condition and the workload at the video servers. In this paper, we propose a mechanism, called adaptive real-time monitoring, to solve the problem. The proposed mechanism consists of three steps: selection, monitoring and switching. In the selection step, the mechanism will determine which video server to serve the client by considering the factors of the current network traffic and workload at the servers. In the monitoring step, the QoS of the video playback will be monitored closely. A feedback mechanism is adopted to improve the QoS of video playback by adjusting the allocation of resources at the video server. The switching mechanism will be invoked when the QoS of the video display is too worse that it must switch to another video server.

Keywords: distributed video player, network jitter, real-time scheduling

1 Introduction

In recent years, a lot of work has been done in the study of distributed video player systems [1, 2, 3, 6]. One of the main issues is how to ensure that the video frames, which are sent from the video server, can arrive and then be displayed on time at the requesting clients so that the display of the video can be maintained at a good quality, e.g. a good quality of services (QoS). Distributed video player systems are firm real-time systems. Compressed video frames, e.g., in MPEG format, are sent continuously from the video server at a pre-defined rate to the

requesting client. Each video frame is associated with a display time defined based on the start time and the play rate of the video. A video frame has to be dropped if it misses its display time [7]. Dropping of video frames can seriously affect the QoS of the video playback and has to be minimized.

A lot of research efforts have been devoted in the design of mechanisms for efficient transmission and compression of video frames especially over an open network environment, e.g., the Internet [1, 3,4]. In such an environment, it is difficult to predict the delay for video frame transmission due to network jitters. The problem is further complicated by the variable sizes of the video frames. Various efficient techniques, such as buffering, software feedback and data steaming, were proposed [1, 3,5] to reduce the impact of the unpredictable network performance on the QoS of video playback. Although these mechanisms have been shown to be effective in improving the system performance, e.g., to provide a better QoS, they may not be suitable to a replicated distributed video player system which consists of several video servers and the same video may be replicated at different video servers [2].

Replicating the same video at different video servers has several important advantages. Firstly, it is obvious that the system will be more fault-tolerant. Secondly, different clients may connect to different video servers even they are requesting the same video. The result is that the workload in the system will be more evenly distributed. Thirdly, a client can choose which video server to connect with so that the transmission delay and overhead for video frame transmission can be minimized.

The effectiveness of video file replication depends heavily on how the mechanism is used to choose the video server for connection, and how the system monitor and maintain the QoS in video playback. Consider a

scenario that a video file is replicated at the video servers in Japan, United States, and Europe, and the client is in Hong Kong. If the selection is simply based on the average transmission delay, the client may choose to connect with the server in Japan since its average transmission delay is the smallest. However, simply consider the average transmission delay is not enough for most of the cases. Other factors, such as the current network connection status, workload of the server and the QoS requirements of the client, are also important factors and should be considered. The actual network delay for individual video frame transmission depends on the current network traffic between the client and server while the frame is transmitting. It can be very large if the traffic workload is heavy at the moment of transmission. Furthermore, the delay will be lengthened if the workload of the server is heavy.

Even though we can determine the “best” video server for connection based on the existing system status, the QoS of the video playback may deteriorate rapidly while it is playing due to the changes in the environment status. (A server is considered to be the *best* if the video is the one, which has the highest probability to give the best QoS for the video playback.) Thus, we need to monitor the status of the display and the system status while a video is playing in order to ensure the QoS of video playback. If the QoS provided to the client drops below a *threshold*, the client may need to switch the connection to another more suitable server.

In this paper, we propose an adaptive real-time monitoring mechanism for replicated distributed video player systems. The objective of the mechanism is to select the “best” video server to serve a client request, and to maintain the QoS while a video is playing. The adaptive real-time monitoring mechanism consists of the three steps: (1) selection of video server, (2) monitor the video playback and (3) switch to another video server in case the QoS of video playback has dropped to an unacceptable level continuously for a certain period of time.

The rest of the paper is organized as follows: Section 2 introduces the adaptive real-time monitoring mechanism. Section 3 discusses the study of the performance of the proposed mechanism using simulation. Section 4 is the conclusions of the paper.

2 Adaptive Real-time Monitoring Mechanism

The proposed mechanism consists of three steps: *optimum server selection (OSS)*, *real-time monitoring* and *dynamic server switching (DSS)*. The optimum server selection step is to select the “best” video server to serve

the client based on the *current status of the network and loading at the video servers*. The real-time monitoring step is to monitor the playback status of a video for a client when the video server is serving more than one client at the same time. It determines how to allocate the resources at the video server to serve the clients. If the display of a video at a client is getting worse and is lower than its requested QoS, the video server may try to rectify the problem by adopting a feedback mechanism in which the priority of the server process, which is servicing the client, will be raised up. It is expected that providing more resource to serve the client, the display status at the client can be improved. The dynamic server switching step is invoked when the QoS of the displayed video is consistently lower than a pre-defined threshold value for a certain period of time even with the use of the feedback mechanism. The client will then perform the server selection mechanism again to select another server to connect with. For this case, it will perform a server switching step, which will be discussed in section 2.4.

2.1 System Components

To support the adaptive real-time monitoring mechanism, one of the video servers is designed as the video server manager (VSM), which maintains a directory of the locations of the video files, e.g., a list containing all the addresses (e.g., the IP address) of the video servers and their video files as shown in Figure 1. The VSM is also responsible for assigning video servers to serve video requests from the clients. When a client requests the playback of a video, it sends a request to the VSM. Then, the VSM sends all the addresses of video servers to the client. The video selection process will then start by performing the optimal server selection step, which is performed by the server processes, the server adaptive controller (SAC) and video packet controller (VPC).

Upon the receipt of a client request, the SAC coordinates with the client to select a video server to serve the client’s request. Once a video server is selected, the SAC and VPC of the selected video server will coordinate with the client and take up the task to serve the client. The VPC of the selected video server controls the rate and actions (e.g. Play, Stop, etc.) by performing the real-time monitoring mechanism. The SAC will perform the dynamic server switching actions in case the QoS of the displayed video is very poor.

Each client contains a client adaptive controller (CAC), which monitors the inter-arrival time of the decompressed frames. If the inter-arrival time drops to an unacceptable value, the CAC will inform the SAC of the video server and it will start the switching server mechanism.

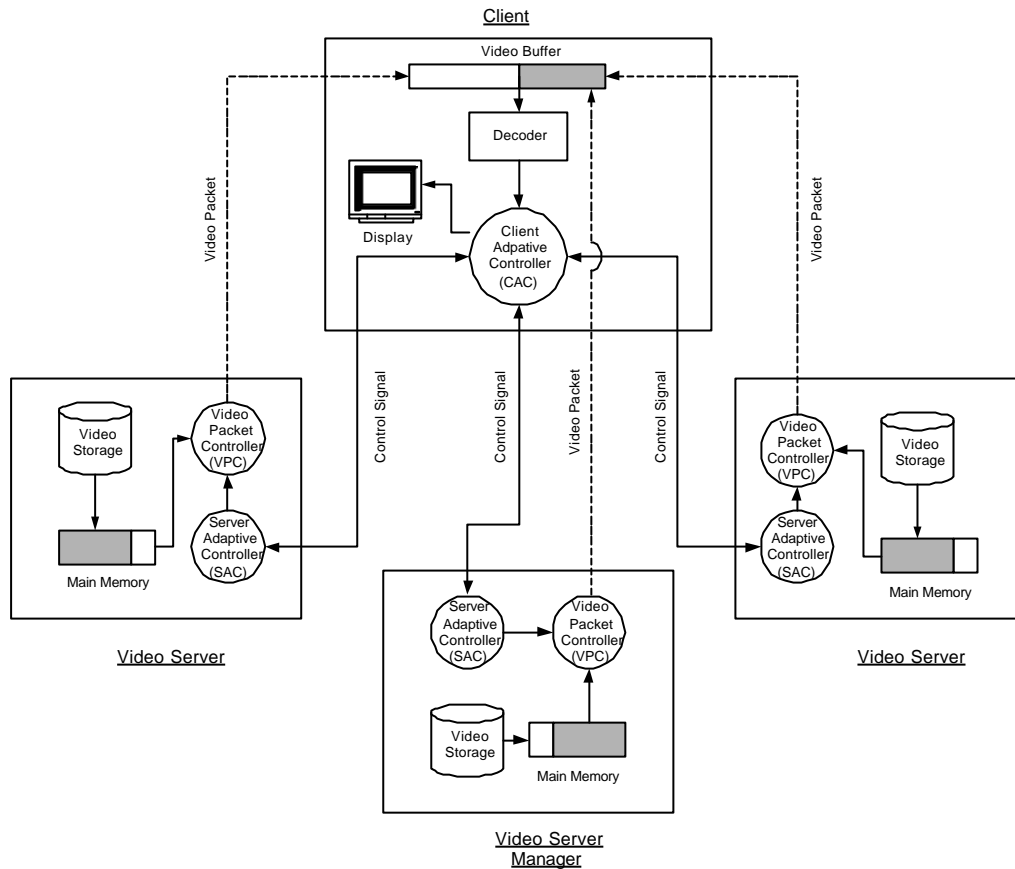


Figure 1: The system architecture for the Adaptive Real-time Monitoring Mechanism

2.2 Optimum Server Selection (OSS)

Since the status of the video servers and the quality of the network connecting the client and different servers can be very different, the connection of a client with different servers can result in a very different QoS of video playback. At the same time, different new connections will have different impacts on the distribution of workload in the system. The purpose of the optimum server selection step is to choose the “best” video server based on the existing system status so that it will give the least impacts on the whole system loading and the best QoS of video playback can be provided to the client.

Before the determination of connecting to which video server, the client will gather information and compute the service workload index (PI) for each server. PI is calculated based on:

1. the server loading (SL);
2. Connected network status (NS); and
3. File workload index (FWI)

Server loading is an index, which indicates the existing workload of the server. It is defined based on the utilization of the video server such as:

$$SL = \text{utilization of video server}$$

The purpose of considering SL is to distribute the workload evenly amongst the video servers.

Connected network status is an index to indicate the overhead for transmitting a video frame from the video server to the requesting client. It can be defined as:

$$NS = \text{round trip delay time from client to server}$$

The value of NS for a video server can be obtained by sending testing messages to the video server.

The file workload index (FWI) indicates the workload characteristics of a video file. A larger value for the file workload index indicates that the workload for transmitting the frames of the video file will be heavier. Different video files will have different characteristics and different file workload index. For example, in the news report program,

there is usually a commentator speaking in front of a still background. This property favors the compression scheme and consumes less resource of the system. Thus, the file workload index will be low. On the other hand, in TV commercials, there are a lot of camera break and the video content switches back and forth rapidly. This property does not favor the compression scheme and consume more resource of the system. Thus, the file workload index will be high indicating that it requires a longer time and more resources to transmit the video frames. The file workload index for a video file is pre-defined and maintained at the video servers.

Now, we define the performance index (PI) for a video server based on *server loading (SL)*, *connected network status (NS)* and *file workload index (FWI)* as:

$$PI = (j \times SL) + (k \times NS) + (h \times FWI)$$

where j, k, h are constants

The followings defines the details of optimal server selection step:

1. Client sends signal to video server manager (VSM) for video play request.
2. The VSM sends the FWI and the addresses of all video servers, which contains the requesting video file, to the client. The FWI for is video file is pre-defined based on its characteristics.
3. Client sends request signals to the video servers for their SLs. (The addresses are obtained from the VSM.)
4. Then it begins to take the start time of round trip delay to calculate the NS.
5. Each video server (VS), which has received requests from the client, sends its SL to the client.
7. Client waits for all the server responses.
8. The client stops waiting until the response time $> D_{max}$, unless there are no server response. (where D_{max} is specified as the QoS of the client for maximum network delay)
9. If there is no server response within the time D_{max} , the corresponding next earliest response server will be the selected server.
10. Otherwise, the client computes the PI of all the servers, which can response with the time D_{max} .
11. The corresponding server of the smallest PI will be the selected server.
12. Client sends a play video request to the selected server.
13. The video playback will start after the selected server receives the play request.

2.3 Real-time Monitoring

A video server may serve more than one client at the same time. Thus, it may have several server processes running concurrently and each server process serves one client. The changing workload at the video server will affect the QoS of the video playback. To maintain the QoS, the status of the video display has to be monitored closely. If the QoS drops below the required QoS of the client, some rectification actions have to be done. If the cause of the problem is due to heavy workload at the video server, we may solve the problem by performing the priority feedback mechanism, which adjusts the allocation of resources at the server to the server processes.

2.3.1 Feedback Control

In servicing the clients, treating all the clients equally is not a good approach to maintain the status of the video playbacks, which are serving by the video server. A better approach is to use a *feedback control* in which a higher priority is assigned to the server process, which is serving a client at a poorer status [1,3]. By adjusting the priorities of the server processes dynamically, the CPU at the video server can be made more “responsive” to the changing status of the clients. Consequently, it is expected that QoS of the video playback can be better monitored.

In the design of feedback control, we have to determine what are the level of control and the control variables. Basically, a distributed video player system can be divided into various components with different levels. Therefore, different levels of feedback mechanisms can be defined in the system for the control of different components. However, the use of multi-feedback mechanisms will incur a heavy overhead. Thus, we choose to apply the feedback mechanism in the highest level, between the server processes and clients. The chosen control variable is the QoS provided to the clients. For simplicity, we choose the number of frames being dropped within a fixed period of time as the control variable. If this number is larger, the status of the client is poorer and it should receive “more services” from the video server than the clients, which are dropping lesser frames. The number of frames being dropped in a period of time x is input into a *priority mapping function* which will calculate the priority for the server process, Priority(P):

$$Priority(P) = \text{priority_mapping_function}(\# \text{ of frames being dropped in period } x)$$

The calculation of the number of frames being dropped in a period is based on the feedback streams sent from the clients. A higher priority means that the server process will have higher priority in accessing the resources and the CPU will serve it first.

2.3.2 Priority Mapping Function

An important component of the feedback mechanism is the design of the priority mapping functions. Since different clients may have different QoS requirements in video playback, different mapping functions may be defined to serve the clients [1]. Basically, there are two types of mapping functions:

- (1) *Linear function*;
- (2) *Increasing function*; and
- (3) *Decreasing function*.

In the *Linear* function, the priority mapping of a process is directly proportional to the number of frames being dropped:

$$\text{Priority}(P) = x + \text{number of frames to be dropped in the period}$$

where x is the default priority of the process (e.g., the priority of a process when it is at the normal status)

The problem of the *Linear* Function is that it treats all the clients the same way according to their number of frames being dropped. If the play speed of the clients in a system is not the same, linear priority mapping will favor the higher play speed clients as they drop more frames as compared with the low play speed clients.

In the *Increasing* function, the rate of increase in priority increases with the number of frames being dropped:

$$\text{Priority}(P) = (\text{number of frames to be dropped in the period})^2 / k$$

where k is a constant.

It has the effect of making the priority feedback mechanism less sensitive to the number of frames being dropped if it is small. It is suitable for the clients with high play speeds since they always need to drop frames (dropping a small number of frames is a consider to be normal status to them.)

The *Decreasing* function is the reverse of the increasing function in which the rate of increase in priority decreases with the number of frames being dropped.

$$\text{Priority}(P) = \sqrt{k \times (\text{number of frames to be dropped in the period})}$$

It has the effect of making the priority feedback mechanism highly sensitive to dropping frames if the number of frames being dropped is small. It is suitable for

the clients with low play speed such that they seldom have to drop frames. Dropping even one frame means that their status is very poor, and the server processes have to be raised up to higher priorities. By using different values of k , we can obtain different distributions for the mapping functions and control the intersection point of the curves.

2.4 Dynamic Server Switching

Real-time monitoring will only be effective to solve the problem when it is caused by a heavy workload at the video server. If the problem is caused by prolonged delays in network transmission of the video frames as a result of heavy network traffic between the client and the video sever, simply adjusting the priority of the server process may not improve the QoS of the video playback significantly. In the DSS step, the client may switch the connection to another video server in order to adapt the changes of the PI. However, an important consideration is the overhead for switching. We define a parameter called *switching overhead (SO)*, which calculates the impact of the switching to the performance. It is also the threshold value that we need to perform the dynamic server switching action. When a client finds that the *client current QoS (CCQ)* value is worse than the SO for a fixed period of time, says z , the DSS mechanism will be performed. Switching Overhead is defined as:

$$SO = m \times (\text{Second best P.I of the system})$$

where m is a constant

Client Current QoS for k period is defined as:

$$CCQ = c \times (\text{smoothness of the video for y period})$$

where c is a constant

Details of the dynamic switching mechanism is described as follows:

1. For the initial y period, set the CCQ to zero because we assume the video quality of this period is very good.
2. After the initial period, we start to compute the CCQ.
3. If $CCQ > SO$ then we step into monitor phase in which the optimum server selection mechanism will start again in order to find the most suitable server for the next connection.
4. If the condition “ $CCQ > SO$ ” still exists for z period continuously, we conclude that the client should switch to another server.
5. Client sets up the connection to the new selected server, which is obtained in the monitor phase.

Figure 2 shows the timing diagram for the mechanism.

3 Experiments

3.1 Experiment Setup and Performance Measures

In the experiments, two performance measures are used to indicate the performance of the proposed mechanism: percentage of displayed frames (PD) and the relative improvement of displayed frames (RI). PD is defined as:

$$PD = \frac{\sum_{i=1}^n D_i}{n \times D_{total}}$$

where n is the number of clients. D_i is the number of displayed frames by client i , and D_{total} is the total number of frames in the video file. RI is defined as:

$$RI = \frac{PD - PD'}{PD'}$$

where PD is the percentage of frames displayed using the adaptive real-time monitoring mechanism, and PD' is the percentage of frames displayed in the replicated distributed video player system without the proposed mechanism, e.g., simply choose the video server which has the smallest mean transmission delay.

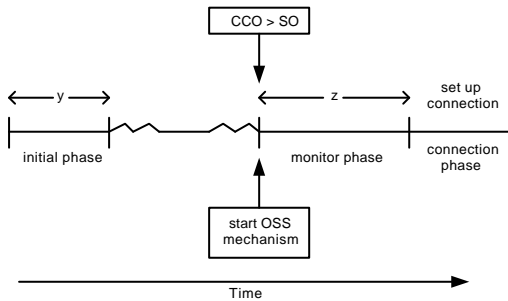


Figure 2 Switching of video server

3.2 Experiment Results

Experiments were concentrated on optimum server selection and dynamic server switching of the proposed mechanism. In the experiments, we study the impact of various factors such as the number of clients, the number of servers and the play speeds on the performance of the proposed mechanisms.

3.2.1 Impact of Number of Clients

In this set of experiments, the clients request video files at a play speed of 25 fps and the video files

requesting by the clients are replicated at 5 video servers. It is assumed that file workload index of the requesting video files are high. The following Table 1 and Table 2 summarize the performance of the proposed mechanism when the number of clients is varied.

From Table 1 and Table 2, we can observe that the system performance (measured in terms of PD and RI) is significantly improved when the OSS and DSS steps are applied. The improvements are more significant when number of clients is greater as a result of higher workload at the video servers and greater network traffic. If the clients choose to connect to video servers that are located closest to them without considering the existing workload of the servers, it may make the distribution of workload in the system uneven. With the use of the OSS, a client may select a server that has the least impact to the whole system performance and this will result in a more even distribution of the workload in the system. Besides, when DSS is switched on, a client may switch to another video server and this mechanism can further improve the system performance. However, since there are some overheads incurred for the switching process, the relative performance only increases slightly as we can be observed by comparing the results shown in Table 2 and Table 3.

# of Clients	Overall PD		Overall Smoothness	
	Normal	OSS + DSS	Normal	OSS + DSS
5	99.000	99.100	0.077	0.070
10	95.300	97.350	0.550	0.375
15	89.160	93.730	1.081	0.575
20	79.250	91.375	2.419	0.763
25	69.980	90.760	3.967	1.024

Table 1: The impact of number of clients on the performance of the OSS mechanism

# of Clients	Overall PD		Overall Smoothness	
	Normal	OSS + DSS	Normal	OSS + DSS
5	99.000	99.500	0.077	0.051
10	95.300	98.200	0.550	0.202
15	89.160	96.267	1.081	0.435
20	79.250	90.775	2.419	0.964
25	69.980	93.040	3.967	0.752

Table 2: The impact of number of clients on the performance of the OSS and DSS mechanisms

3.2.2 Impact of Number of Servers

In this set of experiments, we study the impact of number of servers on the performance of the proposed

mechanism. In the experiments, the number of clients in the system is fixed to be 20.

From Table 3 and Table 4, we can observe that increasing the number of video servers improve the system performance significantly as the workload at the video servers will be lower due to sharing of the workload. Consistent with the results shown in Table 1 and Table 2, the system performance is significantly improved with the application of the OSS and DSS mechanisms. The improvements are more significant when number of servers is smaller. With the use of the OSS mechanism, clients select the video servers that have the least impacts to the whole system. Hence, when there are only a few servers in the system, the loading of the each server will be high. Therefore, there may be a higher probability that many clients may connect to a heavy loaded server. However, with the use of the OSS, the clients can adaptively choose the server so that it will give a minimum impact to the system. On the other hand, in the case when DSS is turned on, since there are very little servers for the clients to choose. There may be very little or even no dynamic switching in DSS and hence the performance is very similar to OSS.

# of servers	Overall PD		Overall Smoothness	
	Normal	With OSS+DSS	Normal	With OSS+DSS
4	99.900	99.100	0.077	0.070
8	95.300	97.350	0.550	0.375
12	89.160	93.730	1.081	0.575
16	79.250	91.375	2.419	0.763
20	69.980	90.760	3.967	1.024

Table 3: The impact of number of servers on the performance of the OSS mechanism

# of server	Overall PD		Overall Smoothness	
	Normal	With OSS+DSS	Normal	With OSS+DSS
4	70.020	89.150	3.175	1.057
8	85.650	97.900	1.759	0.264
12	93.300	97.652	0.760	0.286
16	94.100	98.410	0.310	0.137
20	98.075	99.125	0.189	0.104

Table 4: The impact of number of servers on the performance of the OSS+DSS mechanism

3.2.3 Impacts of Various Play Speeds

From Table 5 and Table 6, we can observe that the system can achieve a much better performance with the use of the OSS and DSS mechanisms. This is consistent with the results shown in previous tables.

Play Speed (fps)	Overall PD		Overall Smoothness	
	Normal	With OSS	Normal	With OSS
22	99.900	99.925	0.0142	0.011
24	92.25	95.825	0.987	0.857
26	71.225	93.450	3.832	0.472
28	30.125	46.875	9.029	5.996
30	5.525	9.650	14.509	12.169

Table 5: The impact of play speed on the performance of the OSS mechanism

The improvements are more significant when the play speed is increased to a higher level (e.g., 26 fps) and then it drops. It can be explained that when the play speed is increasing from small values, the overall loading of the system will become higher. As mentioned in the above, OSS and DSS mechanisms favor to this environment. However, if the play speed is very high, all the cases (normal, OSS and OOS+DSS) cannot endure such a heavy loading. It becomes a saturated state since the performances are poor in all cases at that moment. Hence the system performance cannot be further improved.

Play Speed (fps)	Overall PD		Overall Smoothness	
	Normal	With OSS+DSS	Normal	With OSS+DSS
22	99.900	99.900	0.0142	0.0141
24	92.25	94.325	0.987	0.780
26	71.225	94.325	3.832	0.552
28	30.125	43.225	9.029	6.686
30	5.525	10.425	14.509	11.990

Table 6: The impact of play speed on the performance of the OSS+DSS mechanism

4 Conclusions

In this paper, we propose an adaptive real-time monitoring mechanism for replicated distributed video player system. The mechanism consists of three phases: selection, monitoring and switching.

The optimum server selection step chooses the "best" video server for connection such that this choice has the least impact on the system loading and a smaller overhead for video frame transmission. In the selection, three factors, which are the server loading, connected network status, and file work loading, are considered. The real-time monitoring step is used to monitor the QoS of a video playback while it is playing in order to maintain the QoS of a video playback. The dynamic server switching mechanism is used to retain a continuous good QoS. When the client find that the current QoS is worse than

the switching overhead for a period of time, the client will switch the connection to another server. How to choose this compensated server is based on the optimum server selection mechanism.

The performance of the proposed mechanism is evaluated through simulation experiments. The results show that they can effectiveness improve the system performance in terms of the number of dropped frames and smoothness of the video display.

References

- [1] Kam-yiu Lam, Chris C. H. Ngan, Joseph K. Y. Ng, "Using Software Feedback Mechanism for Distributed MPEG Video Player Systems", *Computer Communications*, vol. 21, pp. 1320-1327, 1998.
- [2] J. F. Koegel Buford, *Multimedia Systems*, ACM Press, New York, 1994.
- [3] Shanwei Cen, Calton Pu and Richard Staehli, "A Distributed Real-time MPEG Video Audio Player", *Proceedings of the 5th International Workshop on Network and Operating System Support of Digital Audio and Video*, April 18-21, 1995.
- [4] Calton Pu and R. Fuhre, "Feedback-Based Scheduling: a Toolbox Approach", *Proceedings of 4th Workshop on Workstation in Operating Systems*, October 14-15, 1993.
- [5] Cripsin Cowan, Shanwei Cen, Jonathan Walpole and Calton Pu, "Adaptive Methods for Distributed Video Presentation", *ACM Computing Surveys*, volume 27, number 4, pp. 580-583.
- [6] A. Vogel, Brigitte Kerherve, Gregor von Bochmann and Jan Gecsei, "Distributed Multimedia and QOS: A Survey", *IEEE Multimedia*, volume 2, number 1, pp. 10-18, 1995.
- [7] Ching-Chih Han, and Kang G. Shin, "Scheduling MPEG-Compressed Video Streams with Firm Deadline Constraints", *Proceeding of the 3rd ACM International Multimedia Conference and Exhibition*, November 1995.