Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

A STUDY OF SELF-ADJUSTING QUALITY OF SERVICE CONTROL SCHEMES

Sheng-Tzong Cheng
Chi-Ming Chen

Computer Science and Information Engineering Dept.
National Cheng Kung University
Tainan, Taiwan

Ing-Ray Chen
Department of Computer Science
Virginia Tech University
Blacksburg, VA, U.S.A.

ABSTRACT

This paper reports simulation methods and results for analyzing a self-adjusting Quality of Service (QoS) control scheme for multimedia/telecommunication systems based on resource reservation. We study the case in which high priority clients' QoS requirement is not changed throughout the service period, while low priority clients' QoS may be adjusted by the system between the maximum and minimum QoS levels specified in order to adapt to the load of the system. The goal of the system design is to optimize the system reward as a result of servicing clients with different QoS and reward/penalty requirements. A QoS manager in these systems can do a table lookup operation using the simulation result reported here to optimize the system total reward dynamically in response to changing workloads during the run time. The simulation result is particularly applicable to multimedia and telecommunication systems in which dynamic QoS negotiation/renegotiation is used as a mechanism to optimize the overall system performance.

1 INTRODUCTION

Quality of Service (QoS) control is an important issue in multimedia/telecommunication systems designed to provide continuous services to clients based on their QoS demands (Oomoto and Tanaka 1993; Oyang et al. 1995; Vina et al. 1994). To date, there are two approaches by which QoS control can be implemented. One approach is based on adaptive, distributed control (Davies et al. 1994; Noble et al. 1995) wherein each client monitors the QoS received and automatically increases or decreases its resource requirement according to actual QoS level delivered to it and also by the amount of resources sensed available in the system. Another approach is based on a priori *resource reservation* (Mercer et al. 1993) wherein a centralized QoS control manager is used to interact with clients. Whenever a client requests a service of the system, it negotiates its QoS requirement with the QoS manager which checks its resources to make sure the client's QoS requirement can be satisfied before admitting the client into the system. In this latter approach, in case there are more clients than the system can handle, or there is a change of system resources, the QoS manager may have to renegotiate with existing clients to lower their QoS requirements so as to meet some performance goals, for example, to increase the number of clients admitted into the system or to decrease the rejection rate. This paper concerns the second approach.

To date, there is not yet a consensus on how QoS reservation, negotiation and renegotiation should be done. Most of the approaches described in the literature are ad hoc and application-specific in nature. One approach is the "deterministic" QoS reservation scheme in which a client is guaranteed of the QoS level negotiated on admission till it terminates. The reservation is normally based on worst-case scenarios. For example, in designing an on-demand multimedia server (Vin et al. 1995), the capacity reservation concept is implemented by allocating a portion of the server capacity to retrieve a specified number of disk blocks in a repeated service cycle for each admitted client so as to meet the playback rate requirements of all admitted clients. A variation to this approach is the "best-effort" or "predictive" QoS reservation scheme in which the client is admitted into the system with QoS guarantee only in a statistical sense (Chang and Zakhor 1996; Vin et al. 1995). In both cases, when the server capacity is used up by existing clients, a newly arriving client is rejected. In these previous studies, the issue of QoS renegotiation was not investigated.

In this paper, we develop a simulation model to analyze a self-adjusting QoS control scheme with considers not only QoS reservation but also QoS negotiation and renegotiation. Furthermore, the QoS renegotiation is initiated by the QoS manager automatically in response to changing client workloads so as to optimize the system

performance dynamically. We also describe a technique to reduce the amount of simulation time to find out the near-optimal condition under which the self-adjusting QoS control scheme can best optimize the performance of the system.

The rest of the paper is organized as follows. Section 2 describes our the system model, including the client workload models, and formulates the QoS control problem as an optimization problem. Section 3 develops a simulation model and discusses how performance data are collected. In addition, it also discusses a technique to reduce the simulation time and how to apply the simulation result to real-time control applications in which the system can dynamically perform QoS reservation and renegotiation in order to optimize system reward in response to changing workloads. Finally, Section 4 concludes the paper and suggests some future works.

2 MODEL ASSUMPTION

We assume that the on-demand multimedia or telecommunication server adopts the *capacity reservation* mechanism (Mercer et al. 1993) such that a QoS negotiation is made at the time a new client arrives. A new client is accepted if (a) the remaining capacity can accommodate the new client based on the negotiated QoS requirement; or (b) the manager can lower the QoS levels of existing clients and make room to accommodate the new client's QoS requirement. Otherwise, the client is rejected. The loss of a client represents a penalty to the system.

We assume that the system consists of a number of QoS slots, each of which corresponds to the minimum amount of resource reservation required to service a client with the lowest QoS requirement. For a video server, for example, the QoS requirement in a slot corresponds to the smallest frame size with black and white display. Naturally, there exists a maximum number of such QoS slots that the system can service without overloading, as having been addressed in previous works in admission control (Chang and Zakhor 1996; Chen and Chen 1996; Chen and Hsi 1998; Vin et al. 1995). Clients with higher QoS requirements must each occupy two or more such slots, e.g., for a video server, this may correspond to a bigger frame size with color video display.

For ease of exposition, we consider a special case when there exist two QoS classes of clients, with each class being characterized by its own arrival/departure rates and reward/penalty values. This assumption of course can be relaxed in the simulation model if desired.

The inter-arrival times of high-priority and low-priority clients requesting for the service of the system can be of any arbitrary distribution, but in the simulation study we assume that they are exponentially distributed with average times of $1/\lambda_h$ and $1/\lambda_l$, respectively. The inter-departure times of high-priority and low-priority clients are also exponentially distributed with average times of $1/\mu_h$ and $1/\mu_l$, respectively.

The system ensures that customers' minimum QoS requirements are satisfied by performing admission control. We classify the clients into high-priority and low-priority categories. We assume that a high priority client specifies a QoS requirement and once the QoS requirement is accepted by the server, it is not to be changed or renegotiated. That is, once a high-priority client is admitted into the system, its QoS must be maintained at the level agreed upon until the client leaves. On the other hand, a low priority client will specify a range of QoS requirements, thus giving the system some leverage to renegotiate its QoS when necessary. The renegotiation can be done in two ways: (a) the system can lower the QoS of low-priority clients in order to accommodate more clients into the system when the resource becomes scarce; (b) the system can raise the QoS of low-priority clients when the resource becomes rich again. Thus, the system can adjust the QoS level of low-priority clients based on the workload to the system, although it must maintain the same QoS level for high-priority clients. QoS guarantee thus applies to high-priority clients while best-effort QoS applies to low-priority clients. This scheme can be extended to several priority classes if needed. This paper addresses only two priority classes. We assume that a high-priority client reserves a fraction $1/n$ of the capacity; a low-priority client also reserves a fraction $1/n$ if the resource is plenty, but gets a fraction $1/m$, $m \geq n$, of the capacity if the resource is scarce upon admission. The system has the leverage to lower or raise the QoS level of a low-priority client with the maximum capacity reservation being $1/n$ and the minimum being $1/m$. While the ratio $m : n$ can be any value, our simulation study will consider the special case in which $m = 2n$, corresponding to the case where the minimum QoS requirement of a low-priority client is exactly one half of its maximum QoS requirement, with the maximum QoS requirement being the same for that of a high-priority client. A low-priority client is thus assumed to have two QoS levels that would allow the system to do QoS control. This restriction is used just to simplify the study; other ratios of $m : n$ can be modeled easily in the simulation program if so desired.

From the perspective of the server system, the system behaves as if it contains N capacity slots. When all slots are used-up, the server can lower the QoS level of low-priority clients, if any is found, to accommodate newly arriving clients, provided that doing so can improve the "pay-off" of the system. The pay-off to the server when a client completes its service is characterized by each client's reward and penalty parameters. The reward/penalty

characteristics of clients are modeled in this simulation study as follows. We assume that the reward which a high-priority client brings to the system is v_h if it is served successfully; on the other hand, the reward which a low-priority client brings to the system depends on the QoS level received: it is v_l during the proportion of the time in which it is being served at the maximum (high) QoS level and v_{ll} during the proportion of the time in which it is being served at the minimum (low) QoS level, with $v_l \geq v_{ll}$. On the flip side, we assume that the penalties to the system when high-priority and low-priority clients are rejected are q_h and q_l, respectively, with $q_h \geq q_l$.

The performance metric being considered in the paper takes both rewards and penalties of clients into consideration. It is called the system's reward rate defined as the average amount of value received by the server per time unit. In other words, under a particular admission policy if the system on average services N_h high-priority clients, N_l low-priority clients with the high QoS level and N_{ll} low-priority clients with the low QoS level per unit time while it rejects M_h high-priority clients and M_l low-priority clients per unit time, then the system's average reward rate is

$$N_h v_h + N_l v_l + N_{ll} v_{ll} - M_h q_h - M_l q_l$$

This reward rate can be translated into the profit rate of a company running the on-demand multimedia/telecommunication service business. The problem that we are interested in solving thus is to identify the best self-adjusting QoS control scheme under which this performance metric is maximized, as a function of model input variables, including N, λ_h, λ_l, μ_h, μ_l, v_h, v_l, v_{ll}, q_h and q_l defined above. Table 1 summarizes the set of model parameters to be used in this simulation study.

Following our earlier work on admission control policies without QoS negotiation control (Chen and Chen 1996; Chen and Hsi 1998), we consider a strategy in which we divide the N slots into three parts: n_h, n_l and n_m, with n_h specifically being allocated to high-priority clients, n_l being allocated to low-priority clients and the remaining n_m slots being sharable to both types of clients. When a high-priority (correspondingly a low-priority) client arrives, if there is a slot available in the n_h (correspondingly n_l) or n_m part, then the client is accepted; otherwise, it is rejected. The policy always fills in the slots in n_h and n_l for high- and low-priority clients, respectively, before filling in a slot in n_m. It can be easily seen that this policy encompasses a special-case baseline scheme in which all slots can be occupied by both types of clients (where $n_h = n_l = 0$). The simulation results later will show that the general scheme at the optimal condition will always perform better than the special-case baseline scheme.

Table 1. Parameters Used in the Simulation Study.

λ_h	arrival rate of high-priority clients
λ_l	arrival rate of low-priority clients
μ_h	departure rate of clients
μ_l	departure rate of clients
v_h	reward of a high-priority client if served successfully
v_l	reward of a low-priority client if served successfully with a high QoS level
v_{ll}	reward of a low-priority client if served successfully with a low QoS level
q_h	penalty of a high-priority client if it is rejected on admission
q_l	penalty of a low-priority client if it is rejected on admission
N	maximum number of server capacity slots for serving clients; one full slot is needed for a high-priority client; one full slot is needed for a low-priority client served with a high QoS level; but only one half slot is needed for a low-priority client served with a low QoS level
n_h	number of slots reserved for high-priority clients only; $0 \leq n_h \leq N$
n_l	number of slots reserved for low-priority clients only; $0 \leq n_l \leq N$ and also $n_h + n_l \leq N$
n_m	number of slots that can be used to service either type of clients; $n_m = N - n_h - n_l$; low-priority clients in this part can do QoS adjustments

3 SIMULATION MODEL AND RESULT

We implemented a discrete-event simulation program to find the best (n_h, n_m, n_l) set under which the system's reward rate is optimized, when given a set of input parameter values. The input to the simulation program is $(N, \lambda_h, \lambda_l, \mu_h, \mu_l, v_h, v_l, v_{ll}, q_h, q_l)$ and the output is the optimal set of (n_h, n_m, n_l) along with the optimal average reward rate obtained by the system, subject to $n_h + n_m + n_l = N$. The simulation program implements the self-adjusting QoS control algorithm as described in Section 2. It maintains an internal structure array of size $2N$, separated into three parts in a ratio proportional to the n_h, n_m and n_l values. For example, if $N = 80$ and $(n_h, n_m, n_l) = (50, 20, 10)$ then these $2N$ slots will be separated into three parts: 100, 40, and 20, meaning that 100 slots in the n_h part will be used by high-priority only (without QoS renegotiation); 40 in the n_m part will be used by either low- and high-priority (with QoS negotiation on low-priority clients); and 20 in the n_l part will be used by low-priority clients only (without QoS renegotiation). Each of these $2N$ slots can accommodate a low-priority client to meet its minimum QoS requirement. A low-priority client can also occupy two slots to meet its maximum QoS requirement. A high-priority client, on the other hand, will always need two slots to meet its QoS requirement.

Each of these slots has a field indicating whether it is occupied or not at any time.

The simulation program is event-driven. Possible events which can cause state changes are given as follows:

1. *Arrival of a low-priority client* - When a low-priority client arrives at the system, the simulation program looks for two free slots in the n_l part. If not available, it looks for two free slots in the n_m part. If neither is found, the low-priority client is rejected; otherwise, the low-priority client is admitted into the system occupying the two slots allocated to it.

2. *Arrival of a high-priority client* - When a high-priority client arrives at the system, the simulation program again examines the internal data structure to see if there are still two slots in the n_h part. If yes, two slots will be allocated to this high-priority client. Otherwise, the simulation program will see if there are two free slots in the n_m part. If yes, the high-priority client will be allocated with the two slots. If neither of the two conditions is true, the simulation program will check if currently there are at least 2 low-priority clients each occupying two slots in the n_m part. If not, the high-priority client is rejected and the simulation statistics is updated; otherwise, the two low-priority clients will each reduce their QoS requirement by occupying only one slot instead of two, thus giving up two free slots in the n_m part to accommodate the high-priority client. This models part of the self-adjusting QoS control capability of the system.

3. *Departure of a low-priority client* - When a low-priority client departs, the resource it occupies is deallocated. It can be in the form of either one slot or two slots being free, depending on the QoS level being allocated to the departing low-priority client at the departure time. Also, because of the deallocation of slots, a low-priority client originally occupying only one slot can increase its QoS level by acquiring another free slot just being released. This models part of the self-adjusting QoS control capability of the system.

4. *Departure of a high-priority client* - When a high-priority client departs, two slots of resources are deallocated. If these two slots are in the n_m part, a low-priority client originally occupying only one slot will increase its QoS requirement by occupying two slots. Thus, a high-priority client's departure in the n_m part can bring two low-priority clients originally running at the minimum QoS level in the n_m part up to the maximum QoS level. This also models part of the self-adjusting QoS control capability of the system.

3.1 Performance Data Collection and Calculation Method

The objective of the simulation program is to collect performance data so as to compute the average reward obtained by the system as a result of executing the self-adjusting QoS control algorithm. Of course, the average reward obtained varies as the (n_h, n_m, n_l) value set changes. The number of (n_h, n_m, n_l) value sets to be tested will be $C(N+2, 2) = (N+2)(N+1)/2$, e.g., for N=32, there will be 527 cases to be tested. The time complexity involved in enumerating and applying the simulation program is thus $O(N^2)$ and it would take a long time to test all the cases before the optimal value set of (n_h, n_m, n_l) is found for a reasonably large N. In the following, we first discuss how we obtain the average reward for a selected (n_h, n_m, n_l) value set. Then, we discuss how we use a search technique to reduce the complexity to get a reasonable near-optimal (n_h, n_m, n_l) value set in $O(N)$.

3.1.1 System Reward for a Given Value Set

For a selected (n_h, n_m, n_l) value set, we compute the average reward rate obtained by the system due to the self-adjusting QoS control algorithm by the *batch means* method. Under this method, the simulation program is executed for a long run divided into batches. A sample mean is computed in each batch. Using these batch means, we then compute the grand mean and the confidence interval. During a batch run, we compute the accumulated reward as a rejection or a departure occurs. A rejected high-priority (low-priority) client takes q_h (q_l respectively) of reward away. A completed high-priority client adds v_h of reward. For a low-priority client, we keep track of the proportion of time it is being served with the high (low) QoS level. If a low-priority client had been served with x time units with high QoS and y time units with low QoS when it departed, then the reward added to the system is $v_l \times x/(x+y) + v_{ll} \times y/(x+y)$. At the end of each batch run, we divide the accumulated reward by the batch run period to get the mean average reward rate for that batch run. A sufficient number of batches are run in the simulation study to make sure that the grand average reward rate obtained has an accuracy of 5 percent at a 95 percent confidence level.

3.1.2 Searching for the Best Value Set

We adopt the *nearest neighbor* search algorithm in order to reduce the time complexity to find the best (n_h, n_m, n_l)

value set under which the system reward is optimized. This approach yields a near-optimal solution, but as we shall see later the result is very close to that obtained by the optimal solution which requires exhaustive search. The idea is to first fix one value among n_h, n_m and n_l, after which we fix one of the remaining two. We adopt the following simple heuristic: if $\lambda_l > \lambda_h$, then fix n_l first; else we fix n_h first. The rationale is that n_l would be important if the arrival rate of low-priority clients is larger than that of high-priority clients since most of the reward generated is likely to be due to low-priority clients. The simulation program is still driven one at a time by a selected (n_h, n_m, n_l) value set. Suppose that n_l is to be determined first. Instead of trying every possible combination of (n_h, n_m, n_l), only n_l varies first to take on all possible values in the range $(0, N)$ one at a time. During a simulation run while n_l is tested at a particular value, n_h and n_m are set to one half of $N - n_l$. e.g., when $n_l = 16$ for $N = 32$, then $n_h = n_m = 8$ in a test run. The best reward value yielded in the $N + 1$ runs with n_l value varying from 0 to N will fix n_l in this case. Then, n_h will vary in the range of $(0, N - n_l)$ with $n_m = N - n_l - n_h$ to see if the reward can be further improved. This method effectively reduces the time complexity in driving the simulation program down from $O(N^2)$ to $O(N)$ at the expense of some solution accuracy. However, as we shall see later this simple approach yields results which are very close to those generated by the optimal solution.

3.2 Simulation Result

The utility of the simulation result can be illustrated with the design of an on-demand multimedia server (Vin et al. 1995) in which it was discovered that the maximum number of client requests that can be served concurrently is $N = 16$ if *deterministic* admission control is considered. It should be mentioned that the number $N = 16$ was obtained based on resource capacity limitations only. Neither the importance of requests nor the QoS negotiation/renegotiation control was considered in (Vin et al. 1995). Below, we study $N = 16$ and $N = 32$ to illustrate the applicability of our simulation model and the proposed self-adjusting QoS control scheme.

Tables 2 and 3 list the optimal (n_h, n_m, n_l) value sets with respect to some selected sets of input model parameter values characterizing various client workload possibilities to the server system for $N = 16$ and $N = 32$, respectively. Table 2 is generated by applying exhaustive search since $N = 16$ is a small number, i.e., by running the simulation program for all possible combination of (n_h, n_m, n_l) value sets and selecting the one that optimizes the average reward rate. In Table 2, we also compare the optimal reward rate with the one obtained by a baseline scheme where $n_h = n_l = 0$, that is, all high and low-priority clients compete for the free slots in the system, although the same self-adjusting QoS control scheme still applies to low-priority clients. In this baseline scheme, since slots are not reserved, high-priority clients can be rejected when the arrival rate of low-priority clients is high relative to the arrival rate of high-priority clients. The reason is that most of the slots may be occupied by low-priority clients, even though low-priority clients can still lower their QoS levels to make room for high-priority clients.

Table 3 is generated by applying both the exhaustive (columns 4 and 5) and the nearest neighbor (columns 2 and 3) search algorithms. It demonstrates that the approximate solutions obtained based on the nearest neighbor search technique are fairly accurate compared with the exact solutions obtained via exhaustive search. In both Tables 2 and 3, we can observe two results. First, the self-adjusting QoS control algorithm at the optimal point will always yield a better reward rate than the special-case baseline scheme. Second, as the system becomes more heavily loaded, the effect of proper QoS reservation and control becomes more significant, as evidenced from the larger differences in reward rate between the optimal case and the baseline case as the client arrival rate increases. The result shows that a QoS reservation and control algorithm such as the one proposed in the paper is important for systems designed to optimize the system overall reward.

4 SUMMARY

Dynamic adjustment of QoS in response to workload changes at the run time is a key element to meet application performance goals. In this paper, we suggested using a uniform performance metric based on the concept of reward optimization as a basis for designing QoS reservation and negotiation/renegotiation algorithms. A self-adjusting QoS control algorithm which will automatically adjust the QoS level of low-priority clients in order to optimize the system total reward has been proposed and examined using a simulation model. The simulation results demonstrated that there exists an optimal way of reserving resources for prioritized clients while the system performs the self-adjusting QoS control scheme on low-priority clients. The optimal condition depends on the input parameter values. One way to apply the simulation result obtained is to statically generate a table covering some perceivable combination of client arrival/departure rates and then do a table lookup at the run time to dynamically perform QoS reservation and QoS negotiation/renegotiation functions so that the system can always optimize its reward rate in response to changing workloads.

A future research area is study more sophisticated QoS control policies such as those in which clients may

Table 1: Optimizing (n_h, n_m, n_l) Set under the Self-Adjusting QoS Control Scheme ($N = 16$)

$(\lambda_h, \lambda_l, \mu_h, \mu_l, v_h, v_l, v_{ll}, q_h, q_l)$	optimal (n_h, n_m, n_l)	optimal scheme reward rate	baseline scheme reward rate
(1, 10, 1, 1, 5, 1, 0.5, 2, 1)	(1,6,9)	14.19	14.17
(1, 10, 1, 1, 10, 1, 0.5, 2, 1)	(3,4,9)	19.72	19.17
(5, 10, 1, 1, 5, 1, 0.5, 2, 1)	(8,8,0)	34.32	31.04
(5, 10, 1, 1, 10, 1, 0.5, 2, 1)	(9,7,0)	69.41	56.01
(10, 10, 1, 1, 5, 1, 0.5, 2, 1)	(8,8,0)	49.64	46.17
(10, 10, 1, 1, 10, 1, 0.5, 2, 1)	(9,7,0)	105.87	92.89
(10, 20, 1, 1, 5, 1, 0.5, 2, 1)	(7,9,0)	41.52	37.60
(10, 20, 1, 1, 10, 1, 0.5, 2, 1)	(8,8,0)	99.39	83.88

Table 2: Optimizing (n_h, n_m, n_l) Set under the Self-Adjusting QoS Control Scheme ($N = 32$)

$(\lambda_h, \lambda_l, \mu_h, \mu_l, v_h, v_l, v_{ll}, q_h, q_l)$	approximate (n_h, n_m, n_l)	approximate reward rate	optimal (n_h, n_m, n_l)	optimal reward rate	baseline reward rate
(1, 10, 1, 1, 5, 1, 0.5, 2, 1)	(8,2,22)	15.00	(10,2,20)	15.00	15.00
(1, 10, 1, 1, 10, 1, 0.5, 2, 1)	(14,2,16)	20.08	(16,3,13)	20.12	20.00
(5, 10, 1, 1, 5, 1, 0.5, 2, 1)	(22,10,0)	36.04	(20,7,5)	36.68	35.00
(5, 10, 1, 1, 10, 1, 0.5, 2, 1)	(24,8,0)	70.97	(24,8,0)	70.97	59.99
(10, 10, 1, 1, 5, 1, 0.5, 2, 1)	(19,10,3)	65.04	(22,10,0)	67.08	59.90
(10, 10, 1, 1, 10, 1, 0.5, 2, 1)	(21,11,0)	130.87	(23,9,0)	134.05	109.90

have several levels of QoS requirements with the triggering conditions depending on the state of the system, such as the load index, percentage of service time of a job, etc. We plan to further refine our simulation model to address these issues.

REFERENCES

E. Chang and A. Zakhor, "Cost analysis for VBR video servers," IEEE Multimedia, Winter 1996, pp. 56-71.

I.R. Chen and C.M. Chen, "Threshold-based admission control policies for multimedia server," *The Computer Journal*, Vol. 39, No. 9, 1996, pp. 757-766.

I.R. Chen and T. H. Hsi, "Performance analysis of admission control algorithms based on reward optimization for real-time multimedia servers," *Performance Evaluation,*, Vol. 33, No. 2, 1998, pp. 89-112.

N. Davies, G.S. Blair, K. Cheverst and B. Friday, "Supporting adaptive services in a heterogeneous mobile environment," *1994 Workshop on Mobile Computing Systems and Applications*, Santa Cruz, California, December, 1994.

C.W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: Operating system support for multimedia applications," *1st IEEE Inter. Conf. on Multimedia Computing and Systems*, Boston, 1994, pp. 90-99.

B.D. Noble, M. Price, and M. Satyanarayanan, "A programming interface for application-aware adaptation in mobile computing," *2nd USENIX Symp. Mobile and Location Independent Computing*, Michigan, April 1995.

E. Oomoto and K. Tanaka, "OVID: Design and implementation of a video-object database system," *IEEE Trans. Know. and Data Eng.*, Vol. 5, No. 4, 1993, pp. 629-643.

Y.J. Oyang, C.H. Wen, C.Y. Cheng, M.H. Lee and J.T. Li, "A multimedia storage system for on-demand playback," *IEEE Trans. Consumer Electronics*, Vol. 41, No. 1, Feb. 1995, pp. 53-64.

A. Vina, J.L. Lerida, A. Molano and D. del Val, "Real-time multimedia systems," *13th IEEE Symp. Mass Storage Systems*, 1994, pp. 77-83.

H.M. Vin, A. Goyal and P. Goyal, "Algorithms for designing multimedia servers," *Computer Communications*, Vol. 18, No. 3, 1995, pp. 192-203.