Adaptive Disk Scheduling Algorithms for Video Servers *

Wonjun Lee and Jaideep Srivastava

Department of Computer Science & Engineering
University of Minnesota at Minneapolis
Minneapolis, MN 55455
{WJLEE|SRIVASTA}@CS.UMN.EDU

Won-Ho Lee

Department of Electronic Engineering
Korea University
Seoul, Korea
LWH@DALI.KOREA.AC.KR

Abstract

Soft-real time applications, such as continuous media (CM) *systems, have an important property, namely, they allow for graceful adaptation of the application Quality-of-Service (QoS), and therefore are able to have acceptable performance with reduced resource utilization. This can be used by the admission control process to decide if an application can be admitted, even if the resource is congested. In this paper, we present a Soft-QoS framework for Continuous Media servers, which provides a dynamic and adaptive admission control and scheduling algorithm. Using our policy, we could increase the number of simultaneously running clients that could be supported and could ensure a good response ratio and better resource utilization under heavy traffic requirements. The observations and findings from the model are validated with simulation studies.*

Keywords: Admission Control and Scheduling, Quality of Service, Video Server, Continuous Media

1 Introduction

1.1 Motivation

The goal of conventional disk scheduling policies is to reduce the cost of seek operations and to achieve a high throughput, while providing fair access to every process that seeks its services. In contrast, the goal of disk scheduling for continuous media (CM) is to meet the deadlines of the periodic I/O requests generated by the stream manager to meet rate requirements. An additional goal is to minimize buffer requirements. In order to ensure continuous and stringent real-time constraints of video delivery in CM servers [4, 13], several factors such as disk bandwidth, buffer capacity, network bandwidth, etc., should be considered carefully and should be handled efficiently. The reservations of these resource factors are required for supporting an acceptable level of display quality and for providing on-time delivery constraints. In particular, disk bandwidth constraint may be the most important factor, given that the I/O bandwidth reserved for each stream on disk depends on latency overhead time, transfer time, defined cycle length, and contention of multiple streams. Hence, we should be able to guarantee that the request of each stream can be fairly supported with good disk utilization and server cost-performance. A video stream being viewed requires timely delivery of data, but it is able to tolerate some loss of the data for small amounts of time. Thus, an acceptable method of degrading service quality is simply to reduce the requested resources of CM applications. The idea of achieving higher utilization by introducing *soft-QoS* is not novel in the networking field. There are several references [2, 11] that study soft-QoS provision in

*This work is supported by Army Research Lab number DA/DAKF11-98-9-0359 to the University of Minnesota.

broadband networks for video traffic and its impact on network utilization. The problem here is dual (i.e., admission control and disk I/O bandwidth management in CM server systems) to that of network call admission control and dynamic bandwidth management.

To handle these issues in CM server systems, we propose a dynamic and adaptive admission control strategy which achieves better performance than the conventional greedy admission control strategies generally used for CM servers. It recognizes that CM (e.g., video) applications can tolerate certain variations on QoS parameters. It develops an algorithm for sharing processing resources at the server to share available resources effectively among contending streams. The proposed algorithm provisions are for *reclamation* (i.e., scheduler-initiated negotiation) to reallocate resources among streams to improve overall the QoS.

1.2 Research Contributions

According to the evaluation of a human's perception on video and audio [20], up to 23% of aggregate video loss and 21% of aggregate audio loss are tolerable. The acceptable values for consecutive loss of both video and audio are approximately 2 Logical Data Units (LDU). Up to about 20% of video and 7% of audio rate variations are tolerable. Henceforth, using this fact, we could *restrict and steal* some resources of running streams, and give them to the other new input requests, based on the logged information and the status of remaining resources given by our admission control algorithm, without causing any perceptible changes for users. It is a reasonable requirement for the CM server to guarantee the given QoS parameters. When the load of CM server is low, it is possible to meet this requirement. But when the number of concurrent CM streams increases in the CM server, it becomes difficult to guarantee all the QoS parameters. Given certain resources, we want to support as many CM streams whose QoS parameters are all acceptable as possible. Furthermore, as more and more clients require CM streams, the QoS of CM server will degrade. It is a good choice to make it degrade gracefully. To guarantee that each client is served with some reasonable quality, admission control is also necessary.

In this paper, we present a dynamic/adaptive admission control strategy for providing a fair scheduling and better performance for video streaming. It efficiently services multiple clients simultaneously and satisfies the real-time requirement for continuous delivery of video streaming at specified bandwidths in distributed environments. This new stream scheduler provides a proficient admission control functionality which can optimize disk utilization with a good response ratio for the requests of clients; in particular, under heavy loaded traffic environments. It still presents similar levels of stream qualities (how well the requested rate is attained in CM servers) compared to the basic greedy admission control algorithm. We will present the comparison of the simulation result on the behavior of conventional greedy admission control mechanisms with that of our admission control and scheduling algorithm.

The rest of this paper is organized as follows. In Section 2, we briefly describe an assumed CM server architecture and the admission control constraints (I/O bandwidth requirement and buffer constraint). The detailed description of our admission control and scheduling algorithm is presented in Section 3. Section 4 gives the quantitative results of experimental evaluations, and describes experimental designs and parameters. Section 5 describes the related work, and finally concluding remarks and on-going work appear in Section 6.

2 Architecture of CM Server

The CM server system is a typical *client/server* application. It is based on several file systems: for example, the *Presto File System (PFS)* [10] and the *Unix file system (UFS)*. The core part of the *CM server* involves the *network manager*, the *QoS manager*, *I/O managers*, and *proxy servers*. These sub-modules in the CM server are threads running in a single process More details of the architecture of CM server system where the ad-

mission decision is made, can be found in [7, 9]. The admission control for input CM streams is managed by the *QoS Manager*, which is composed of the *Admission Controller* and the *QoS Handler*. The Admission Controller provides two constraint tests: (1) I/O bandwidth test; and (2) available buffer test. Each request (stream) arrives with some *rate* value (e.g., playing-back rate : i.e., frames per second) and the Admission Controller determines whether to admit it or not. The QoS Handler takes care of data rate handling according to a rate input parameter provided by the client's request. In case the playing time gets delayed due to some kind of system overhead, this module will drop some frames properly so as to keep the data rate.

2.1 Basic Conditions of Admission Control

Every time either (1) a new client's request arrives; (2) a rate control operation (such as *Set_Rate, Pause, Resume,* and *Fast_Forward*) is received; (3) the playing of a running stream is over; or (4) when the resources in the reserves are not used for some time, the following constraint must be checked for admission control requirement.

$$\left(t_s + \sum_{i=1}^{n+1} \lceil \frac{T_{svc} \cdot r_i}{b} \rceil \cdot t_r \right) + \sum_{i=1}^{n+1} \frac{T_{svc} \cdot r_i}{R} \leq T_{svc} \quad (1)$$

i.e., latency overhead time(= seek time + rotational latency) + transfer time $\leq$ cycle length

where, t_s: seek time (msec); T_{svc}: cycle length (msec); r_i: consumption rate (Byte/msec); b: disk block size (Byte); t_r: rotation time (msec); and R: transfer rate (Byte/msec).

The first element (t_s) explains that the maximum disk seek latency overhead in a cycle. The second component shows the sum of each rotational latency incurred for retrieving each disk block (for each request r_i) in a given cycle length (T_{svc}). The last component describes the time to transfer r_i during a cycle. Finally, the overall time for disk access time in a given cycle length must be done in the service cycle length (T_{svc}). Since we only consider the I/O bandwidth constraint in this paper, we do not present the details of the constraint of buffer requirement here.

3 Adaptive Disk Scheduling

Most of the existing admission control approaches are purely greedy strategy in the sense that a new application (video stream) can be accepted only if the server could give the client all the requested resources [1, 18]. These approaches are too conservative and admit too few streams, thereby under-utilizing the server resources. Although probabilistic methods exist to amortize the cost of this failure, this is undesirable in general [12].

We propose an enhanced admission control algorithm ($\mathcal{RAC}$: *Reserve-based Admission Control Algorithm*) in that it is capable of readjusting resources according to the amount of remaining resources. The key idea here is to assign a portion of the resources as the reserves, and when the applications start to dip into the reserves, another strategy is invoked. In heavy loaded traffic (when a fairly large number of client requests want services; for example, the total disk bandwidth utilization gets over 70%), if the remaining available resources become smaller than some value (**threshold:** $T_{reserve}$), we assign only some portion of the requested resources to the new requests according to the **done_ratio** and *available resources*. For the degraded streams to adapt based on the availability of resource, **resource-negotiation** is required. *Resource-negotiation (reclamation)* occurs under these circumstances: (1) when a running request returns the resources back to the system; (2) when *set_rate* video functions, such as *Fast_Forward*, are received, or (3) after some period elapses without any further resources being used. Table 1 describes the attributes used in the algorithms.

3.1 $\mathcal{RAC}$ Algorithm

The key advantage of the basic greedy strategy is that it is simple. However, due to the other shortcomings of basic greedy admission control

Attributes	Descriptions
s_i	stream i
q_i	initially requested resource of s_i
v_i	currently serviced resource of s_i : $= q_i \star d_i$
$\mathcal{RS}$	set of all the running streams : $\bigcup_i s_i$, where $\{s_i \mid F_{min,i} < d_i \leq 1.0\}$
$\mathcal{DS}$	set of degraded streams : $\mathcal{RS} - \bigcup_i s_i$, where $\{s_i \mid d_i = 1.0\}$
s_{min}	stream j of which d_j is smallest in $\mathcal{DS}$: $s_{min} = \{s_j \mid \forall_{s_j, s_k \in \mathcal{DS}} d_j \leq d_k\}$
m	number of streams in $\mathcal{RS}$
done_ratio	ratio of serviced resource to requested resource for a stream
remaining_ratio	ratio of resource yet serviced to requested resource for a stream: $(= 1.0 - done\_ratio)$
$remaining\_rate_i$	amount of resource which is yet serviced in stream i : $(= q_i - v_i)$
sum_rem_ratio	total sum of remaining_ratio of degraded streams : $\sum_k e_k$, where $s_k \in \mathcal{DS}$
MIN_FRACT	minimum amount of resource to be assigned for a stream
MIN_RESERVE	minimum amount of reserve which must be at least maintained in $\mathcal{DRA}$ mode
d_i	done_ratio of s_i
e_i	remaining_ratio of s_i : $(= 1.0 - d_i)$
$F_{min,i}$	MIN_FRACT of s_i
T_{total}	total available resources initially given
$T_{reserve}$	amount of resources assigned to the reserve
T_{free_res}	available resources in Reserve
$T_{free\_non\_res}$	available resources outside Reserve
T_{alloc}	allocated resources to the requesting stream
T_{avail}	available resources to assign: $(= T_{total} - T_{used})$
T_{used}	total resources currently used : $\sum_i v_i$, where $s_i \in \mathcal{RS}$
$\Psi(T_{free\_res})$	heuristic function for reserve assignment in *Admission Test* : e.g. $\frac{T_{free\_res}}{k}$, where $k = k_1^{(T_{used} - T_{reserve}) \cdot k_2}$
$\Phi(T_{free\_res})$	heuristic function for reserve assignment on *Close of streams* : e.g. $\frac{T_{free\_res}}{k}$, where $k = k_3 \cdot (T_{used} - T_{reserve}) + 1$
congestBit	bit flag to indicate congested state : $\begin{cases} 1 & \text{if } T_{used} > T_{total} - T_{reserve} \\ 0 & \text{otherwise} \end{cases}$
$\mathcal{SU}(t)$	total system utilization defined by $\sum_k d_k$, where $\forall_k s_k \in \mathcal{RS}$ at time t
$\mathcal{QT}(t)$	total video quality defined by $\mathcal{SU}(t)/m$ at time t

Table 1: Attributes Used in Algorithm

algorithms, we should be able to think of another strategy to allow more streams to run concurrently in the continuous media servers by degrading the requested quality of newly arriving streams and by adapting the returned resources to these streams. Given the average seek time (t_s), cycle length (T_{svc}), disk block size (b), rotation time (t_r), and disk transfer rate (R), the consumption rate of each client request (q_i) is a variable in the I/O bandwidth constraint equation (Eq. 1). We initialize the total available rate (T_{avail}) with $T_{svc}(= T_{total})$, and set $T_{reserve}$ (the **threshold** value for criteria to check *congested_bit*). Initially the *congested_bit* is 0 (not congested). The resource (here I/O bandwidth) is **congested** if the resource usage (T_{used}) is more than (T_{svc} - $T_{reserve}$). We can set the $T_{reserve}$ amount of resources for admitting more requests at a reduced quality. Here, $T_{usage} + T_{avail} = T_{svc}$. That is, the *congested_bit* is set $(= 1)$ only when T_{avail} becomes smaller than $T_{reserve}$; otherwise, the basic I/O bandwidth constraint is just applied (i.e., the new request is admitted without being degraded). Under the *congested_bit* being set, we restrict the amount of assignment of resources to the new request because there is no sufficient resource remaining any longer). We explain the details in the following:

1. On adding the request q_i, if the resource remains uncongested, then admit it with no degradation.
   $T_{alloc} = q_i$.

2. Let the current usage be T_{used}; then, adding the application q_i will increase the usage to $(T_{new} = T_{used} + q_i)$. If $(T_{new} \geq T_{svc} - T_{reserve})$; then, adding q_i will make the resource congested. When the resource gets congested, we have to dip into the resources.
   $T_{free\_res} = \max(0, T_{svc} - T_{reserve} - T_{used})$.
   This is the unused resource outside the reserves.
   Let T_{avail} be the amount of resources in the reserves.
   $T_{free\_res} = \min(T_{reserve}, (T_{svc} - T_{used}))$.
   The new application is allocated, and then
   $T_{alloc} = T_{free\_res} + \min(\ (q_i - T_{free\_res}), \Psi(T_{free\_res}))$.
   That is, we allocate the resources that are outside the reserves and, in addition, we give a maximum $\Psi(T_{free\_res})$ of the resources in the reserves. $\Psi(.)$ is a function of T_{free_res} and returns an appropriate amount of resources according to T_{free_res}.

3. If $\Psi(T_{free\_res})$ is too small (in this case, we had better not support the new request because display quality may be too poor), we simply reject the request.

4. If the running stream (s_k) is over, then the resource v_i is reclaimed. We return the resource occupied by the stream and adjust (if required) the resource allocation of streams not being fully serviced. Here the unused resource is allocated to applications such that the least serviced streams could get the returned resource first.

5. **Negotiation (Reclamation) algorithm**: we have implemented two heuristic-based methods ($\mathcal{DRA}$ and $\mathcal{RWR}$) and tested their performance.

6. *Dynamic Reserve Adaptation ($\mathcal{DRA}$)* : this is invoked only if there are applications that need resources and do not have them. On departure of streams, $\mathcal{DRA}$ returns the assigned resources to the leaving streams (q_i) back to the available resource pool and re-calculates the new T_{avail} using the total available resource (i.e. old $T_{avail} + q_i$). Among the requests that are not fully serviced yet, we select a request (s_{min}) from the queue ($\mathcal{DS}$), whose *done_ratio* is the smallest first, and assign the proper resource to the request. The **maximum $\Phi(T_{free\_res})$-rule** applies here.

   Assign T_{alloc} = min ($\Phi(T_{free\_res})$, remaining_rate$_{min}$) to the selected request (s_{min}).
   The loop continues until there is no more available resource to assign or T_{alloc} is too small to assign.

7. *Reclamation within Returned Reserve ($\mathcal{RWR}$)* : this is a similar method to the Dynamic Reserve Adaptation method, but the difference is that it redistributes the only resources returned from the leaving streams (v_i). In $\mathcal{DRA}$, we used T_{avail} (instead of v_i) for reclamation. The T_{alloc} per stream is calculated according to the ratio of *remaining_ratio* to *sum_remaining_ratio*.

   T_{alloc} = T_{avail} * ($remaining\_ratio_k / sum\_remaining\_ratio$)
   The key idea here is not to touch the reserves, but to utilize the only returned resource due to the departure of streams. We will validate the better performance of this policy compared to that of $\mathcal{DRA}$.

8. When the resources in the reserves are not used for a long time, some of it is reclaimed. For every k period, if there is no request to the reserve, then we release some amount of the reserve to be reclaimed. This is kind of a *reserve adaptation* method.

We present the algorithm (*Reserve-based Admission Control and Scheduling Algorithm* ($\mathcal{RAC}$)) here.

Algorithm 1 $\mathcal{RAC}$ *($T_{reserve}$, event E_i)*

```
   ▷ Reserves-based Admission Control
   ▷ E_i = {(START, q_i), (CLOSE, v_i)}
1  switch EVENT of
2    case "START" :
3      if (! congested) then
           ▷ Admit the application with no degradation
4          T_alloc ← q_i;
5          T_used ← T_used + q_i;
6      else /* congested */
7          if (Ψ(T_free_res) > MIN_FRACT ) then
               ▷ Admit at a lower quality
8              T_alloc ← min( Ψ(T_free_res), q_i) /* ∈ (0, q_i) */
9              T_used ← T_used + T_alloc;
10             degraded = 1;
11         else
               ▷ too small to admit
12             Reject s_i;
13   case "CLOSE" :
       ▷ with parameters of appl_inst_ID
14     if (degraded)
           ▷ add back resource v_i to our count
15         T_used = T_used − v_i;
16     switch MODE of
17       case "DRA" :
           ▷ resource v_i + T_avail is reclaimed to distribute
             it to the degraded application instances
18         T_avail ← T_total − T_used;
19         while (1) do
20             s_min ← Select a request of which done_ratio
               is the smallest;
21             if (Φ(T_free_res) > MIN_RESERVE) then
22                 T_alloc = min (Φ(T_free_res), remaining_rate_min);
23                 T_used ← T_used - T_alloc;
24         end while
25       case "RWR" :
           ▷ resource v_i is reclaimed to distribute it to
             the degraded application instances
26         T_avail = v_i;
27         sum_remaining_ratio = Σ_k remaining_ratio_k ;
28         for (each degraded application k) do
29             T_alloc = T_avail * (remaining_ratio_k / sum_remaining_ratio);
30             T_used ← T_used − T_alloc;
31         end for
32     end switch
33 end switch
```

4 Experimental Evaluation

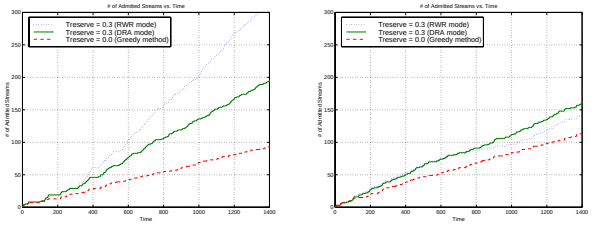
We performed extensive simulation to validate our admission control and scheduling algorithm. In this section, we present the performance results obtained from simulations under the various load conditions. For results presented in this section, we simulated an environment: i.e., a Sun Ultra Sparcstation with a Seagate Barracuda 4GB disk (ST34371N). The details of the simulation parameters can be found in [8].

4.1 Experiment Design

We simulate the situation where the applications with soft-QoS requirements arrive at the system in an arbitrary order. The applications also come with a play-out rate (which maps into an amount of required resource), depending on the user profile and the data content, which is a value of the data rate parameters. The applications arrive based on a Poisson process with an average inter-arrival rate (λ). The duration of the applications is described using a Gaussian process with the mean (μ_d) and the standard deviation (σ_d) of the distribution. The admission control and scheduling algorithm is tried on each one of the application traffic traces. We measure the following: (1) *Accumulated number of admitted streams over time*; (2) *Total system utilization*; (3) *Total quality*; and (4) *Admission ratio*. For our simulation, we made various application load traffic sequences and different application load conditions, and chose the following two distributions: *Heavy traffic* is generated by setting $\lambda = 0.333$, $\mu_d = 90$, and $\sigma_d = 3.5$. *Medium traffic* is generated by having $\lambda = 0.125$, $\mu_d = 60$, and $\sigma_d = 3.5$. The values on the x-axis are normalized. Under the heavy traffic, the resource demands arrive more frequently than under the medium traffic.

4.2 Experiment 1: Number of Admitted Streams

The major advantage of the $\mathcal{RAC}$ is its guaranteeing more numbers of streams to be admitted and to run simultaneously with a tolerable degradation of quality. As shown in Figure 1, under traffic loads that demand more than the available resources, the accumulated occurrences of admission decreases (rejection increases), both in the basic method and in our strategy. The heavier the traffic is, the better performance the $\mathcal{RAC}$ algorithm achieves compared to the basic greedy algorithm (as shown in Figure 1(a)). In another point of view, the $\mathcal{RWR}$ algorithm achieves a 200 $\sim$ 300% increase in the number of streams that can be serviced simultaneously by the server. In case of the $\mathcal{DRA}$ algorithm, it also achieves about a 100% increase. In other words, our $\mathcal{RAC}$ algorithm (both $\mathcal{RWR}$ and $\mathcal{DRA}$) noticeably reduces the rejection ratio compared to the basic greedy method.

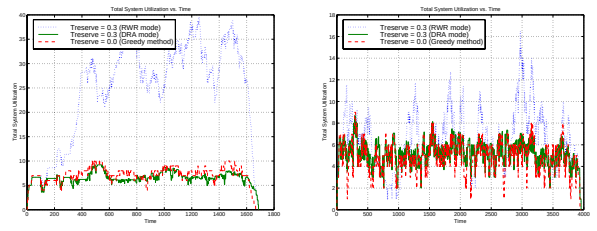


(a) Heavy Traffic (b) Medium Traffic

Figure 1: Number of Admitted Streams over Time

4.3 Experiment 2: System Utilization and Total Quality

In this section, we mainly focus on visualizing the effect of the $\mathcal{RAC}$ algorithm with respect to total system utilization and display quality. Figure 2 plots the total system utilization that is achieved by the sum of each application's *done_ratio*. It is observed that the total system utilization achieved by $\mathcal{RWR}$ is much higher during most of the period. Hence, the $\mathcal{RWR}$ algorithm is able to utilize the system resources more efficiently than the greedy algorithm.

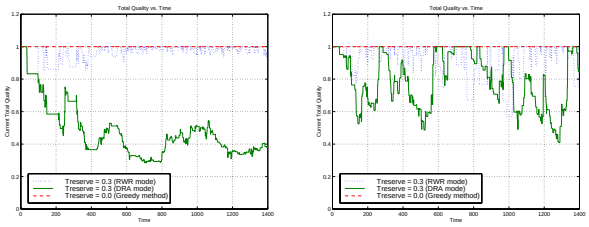


(a) Heavy Traffic (b) Medium Traffic

Figure 2: Total System Utilization over Time

To investigate the display quality of the $\mathcal{RAC}$ algorithm, we plot another curve. Figure 3 illustrates the total quality which is quantified by dividing the sum of each application's done_ratio

by the number of current running streams. It is observed that the total quality achieved by the $\mathcal{RWR}$ algorithm is almost the same as that achieved by the greedy algorithm. In contrast, the performance of $\mathcal{DRA}$ algorithm in total quality is kind of low most of the time. This arises due to the fact that the $\mathcal{DRA}$ algorithm holds off the reserves for future streams, which arrive soon before some of the current running streams depart. Under medium traffic, though the performance of $\mathcal{DRA}$ gets better, it is still too low.

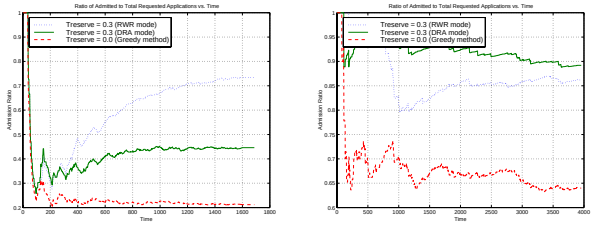


(a) Heavy Traffic (b) Medium Traffic

Figure 3: Total Quality over Time

4.4 Experiment 3: Admission Ratio

Under the heavy load, $\mathcal{RWR}$ achieves the best admission ratio (a 150 $\sim$ 250% increase), and $\mathcal{DRA}$ achieves the next (about 100% increase). Under the medium load, the increase rates of the two $\mathcal{RAC}$ methods are decreased compared to under the heavy load. Unlike the heavy load, the $\mathcal{DRA}$ method achieves even better performance than $\mathcal{RWR}$. This is because when the traffic load is not too heavy, $\mathcal{DRA}$ is able to utilize the resources more efficiently by taking off some resources from the reserves. The probability of reserves running out under medium/light loads is smaller than that under the heavy load. However, we can observe that the $\mathcal{RAC}$ (both $\mathcal{RWR}$ and $\mathcal{DRA}$) strategy noticeably increases the admission ratio compared to the greedy method. Degradation makes the applications use a lower (but fully reasonable due to the properties of CM) amount of resources; therefore, more applications can be supported on the resource.



(a) Heavy Traffic (b) Medium Traffic

Figure 4: Admission Ratio over Time

5 Related Work

Several techniques for the admission controlling of continuous media have been proposed in the literature. Ensuring continuous retrieval of each strand requires that the service time not exceed the minimum of playback durations of the blocks retrieved for each strand during a round, which is a typical greedy admission control strategy [5, 19]. Usually the simple admission control decisions are based on the **worst case** scenario. The worst case policy is based on the observation that multimedia traffic is characterized by its bursty nature; therefore, if sufficient resources are not available for the worst case scenario, then the applications may fail [16]. Similar admission control based on the worst case analysis of the application stream has been formulated by other researchers [14, 15, 3].

To improve the resource utilization, predictive (observation-based) admission control algorithms for clients have been suggested [17]. It simply uses the average amount of time spent in retrieving a media block instead of the worst-case assumptions Vin et al. [18] described a statistical admission control algorithm in which new clients are admitted for service as long as the statistical estimation of the aggregate data rate requirement, rather than the corresponding peak data rate requirement, can be met by the server, They improve the utilization of server resources by exploiting the variation in the disk access times of

media blocks, as well as by exploiting the variation in playback rate requirement by variable rate compression techniques. Statistically they are safe but in the worst case it may cause applications to fail because of resource congestion.

6 Concluding Remarks

We presented an integrated adaptive admission control and scheduling algorithm for CM server systems. We have also presented results of a simulation evaluation of our algorithm and a greedy algorithm with respect to several metrics designed to measure the admission ratio, total quality, and system utilization. It was observed that under heavy traffic, our algorithm achieves much better performance than the greedy algorithm. Using our scheme, we could expect that more streams could be running with an acceptable range of data quality in a given system resource. Our ongoing [6] and future work include extending the QoS metrics considered in our algorithm to multi-dimensional ones (e.g., frame size, buffer requirement, compression quality factor, scalability, network bandwidth, etc.).

References

[1] F. T. Ernst W. Biersack. Statistical Admission Control in Video Servers with Constant Data Length Retrieval of VBR Streams. In *Third International Conference on Multimedia Modeling*, Toulouse, France, Nov 1996.

[2] J. W. G. de Veciana, G. Kesidis. Resource Management in Wide-Area ATM Networks Using Effective Bandwidths. *IEEE Journal on Selected Areas in Communication*, 13(6):1081–1090, Aug 1995.

[3] D. Gemmell and S. Christodoulakis. Principles of Delay Sensitive Multimedia Data Storage and Retrieval. *ACM Trans. Information Systems*, 10(1):51–90, 1992.

[4] R. Haskin and F. Schmuck. The Tiger Shark File System. In *COMPCON 96*, 1996.

[5] S. W. Lau and J. C. S. Lui. A Novel Video-On-Demand Storage Architecture for Supporting Constant Frame Rate with Variable Bit Rate Retrieval. In *5th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Durham, N.H. , 1995.

[6] W. Lee and B. Sabata. Admission Control and QoS Negotiations for Soft-Real Time Applications. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems (ICMCS'99)*, Florence, Italy, June 1999.

[7] W. Lee and J. Srivastava. CORBA Evaluation of Video Streaming wrt QoS Provisioning. In *Proceedings of the Workshop on Multimedia Networking held in conjunction with the 17th IEEE Symposium on Reliable Distributed Systems (SRDS '98)*, West Lafayette, Indiana, October 1998.

[8] W. Lee and J. Srivastava. Dynamic/Adaptive Algorithms for Admission Control and Scheduling of Video Servers. *Submitted for publication in IEEE Transactions on Computers*, 1999.

[9] W. Lee, D. Su, and J. Srivastava. QoS-based Evaluation of File Systems and Distributed System Services for Continuous Media Provisioning. *To appear in Information and Software Technology, Elsevier Science*, 1999.

[10] W. Lee, D. Su, J. Srivastava, D. Kenchammana-hosekote, and D. Wijesekera. Experimental Evaluation of PFS Continuous Media File System. In *6th ACM Int'l Conf. on Information and Knowledge Management (CIKM '97)*, November 1997.

[11] D. M. Grossglauser. Measurement-Based Call Admission Control: Algorithms and Analysis. In *IEEE INFCOM'97*, April 1997.

[12] D. Makaroff, G. Neufeld, and N. Hutchinson. An Evaluation of VBR Disk Admission Algorithms for Continuous media File Servers. In *Proceedings of the ACM Multimedia Conference*, Seattle, Wa, Dec 1997.

[13] C. Martin, P. S. Narayanan, B. Ozden, R. Rastogi, and A. Silberschatz. The Fellini Multimedia Storage Server. In S. M. Chung, editor, *Multimedia Information Sotrage and Management*. Kluwer Academic Publishers, 1996.

[14] P. V. Rangan and H. M. Vin. Designing a Multiuser HDTV Storage Server. *IEEE Journal on Seleted Areas in Communications*, 11(1):153–164, January 1993.

[15] A. Reddy and J. Wyllie. I/O Issues in a Multimedia System. *Computer*, 27(3):69–74, 1994.

[16] S. K. T. S.V.Raghavan. *Networked Multimedia Systems: Concepts, Architecture, and Design*. Prentice Hall, 1998.

[17] H. M. Vin, A. Goyal, , P. Goyal, and A. Goyal. An Observation-Based Approach for Designing Multimedia Servers. In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Boston, MA, May 1994.

[18] H. M. Vin, P. Goyal, A. Goyal, and A. Goyal. A Statistical Admission Control Algorithm for Multimedia Servers. In *Proceedings of ACM Multimedia '94)*, San Francisco, October 1994.

[19] H. M. Vin and P. V. Rangan. Admission Control Algorithms for Multimedia On-Demand Servers. In *3rd International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 1992.

[20] D. Wijesekera and J. Srivastava. Experimental Evaluation of Loss Perception in Continuous Media. *to appear in ACM Springer Multimedia Systems Journal*, 1999.