Threshold-Based Admission Control Policies for Multimedia Servers

ING-RAY CHEN AND CHI-MING CHEN

Institute of Information Engineering, National Cheng Kung University, Tainan, Taiwan, ROC
Email: irchen@iie.ncku.edu.tw

Traditional admission control algorithms for on-demand multimedia servers concern the acceptance decisions for new clients' requests so as to guarantee that continuous services to all clients are executed. These algorithms determine whether a new client can be accepted, based on the consideration whether the underlying hardware can satisfy the quality of service (QoS) requirements of admitted client requests. In this paper, we consider a richer class of admission control algorithms that make acceptance/rejection decisions not only to satisfy the hardware requirements of client requests but also to optimize the reward of the system based on a performance criterion as it services clients of different priority classes. We divide the server capacity into a number of 'priority threshold values' based on which the system decides whether to accept clients of different priority classes dynamically in order to maximize the system value. The resulting threshold-based admission control algorithm is developed based on the idea that admission control can be driven not only by hardware requirements, but also by knowledge regarding the workload characteristics of client requests, thus allowing the system to adjust dynamically the threshold values in response to changes in client workload characteristics. We derive a close-form expression for the value which the system can obtain when operating under the threshold-based algorithm as a function of model parameters, and discuss how the server can utilize the analytical solution at run time so as to maximize the system value dynamically without violating clients' continuity requirements.

Received February 23, 1996; revised February 6, 1997

1. INTRODUCTION

Multimedia servers [1, 2, 3, 4, 5] are designed to provide continuous services to clients on demand. This can be achieved by having the server periodically execute the clients' tasks (e.g. for media data processing) such that the periodic deadline requirement of each client is satisfied. A promising approach for guaranteeing continuous services is based on the design concept of *capacity reservation* [6] by which a fraction of the server capacity is reserved for each new client, and once a reservation has been made, the client is guaranteed of the availability of the server capacity reserved until it terminates. For example, in designing an on-demand multimedia server [4], 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 play-back rate requirements of all admitted clients. When the server capacity is used up by existing clients, a newly arriving client may be rejected so as to guarantee continuous services to clients that have been admitted. The rejected client is then 'lost', representing some type of loss to the system in overloaded situations.

One issue in the design of such on-demand multimedia servers is to make the loss rate of clients as small as possible when the system is overloaded. This issue may involve dynamically lowering the quality of service (QoS) levels of existing clients so as to make room for newly arriving clients and may involve the design of some negotiation protocols [7]. Another design issue is an admission control policy based on which the server accepts/rejects new requests. This paper concerns the second design issue.

Existing admission control algorithms make acceptance/rejection decisions merely based on the consideration whether the underlying hardware can satisfy the QoS requirements of clients [8, 4, 9, 10], e.g. based on the play-back rate requirements of media streams. Two classes of admission control algorithms have so far been considered in the literature, namely, 'deterministic' and 'best-effort.' Rangan *et al.* [4, 9] developed deterministic admission control algorithms by considering different ways of performing data placement and disk access scheduling. They derived a formula for the maximum number of clients which the system can admit based on the theoretical worst-case time bounds with absolute guarantees. Vin *et al.* [10] subsequently compared deterministic algorithms with predictive algorithms which allow momentary violations of the clients' QoS requirements to be tolerated, as long as the fraction of media data delivered on time is larger than a specified threshold value, say α, specified as part of a client's QoS requirement. Based on this concept, they developed predictive admission control algorithms using average-case service times obtained from extrapolation data based on past measurements. They

claimed by simulation that the maximum number of clients which can be admitted by the server with predictive services is greatly increased with the server utilization being greatly improved, when compared with deterministic services. It was also demonstrated that these benefits are obtained without sacrificing the QoS requirements specified by clients requesting predictive services. Another study of best-effort admission control is by Chang and Zakhor [8] who investigated a class of statistical admission control strategies for buffer management for variable bit rate (VBR) video servers. They used a statistical QoS control strategy to determine whether a client can be admitted based on a statistical estimation of the probability that the total data requested by all of the users exceeds the server capacity. A new client to the system is rejected if this probability at the arrival instant is found to be greater than a specified threshold probability value. They showed that statistical admission control is more effective than deterministic admission control for interactive video server systems.

All these past studies discussed above do not consider priority scheduling in operating environments in which multiple service classes exist. Their main research goal was to accept as many clients as possible without violating or compromising too much of their QoS requirements. In this paper, we address priority scheduling issues and develop admission control algorithms which make acceptance/rejection decisions not only to satisfy the QoS requirements of client requests but also to make the system benefit the most from the perspective of 'reward optimization'. We borrow the concept of transaction values in scheduling real-time transactions[1] [11, 12] and introduce the notions of 'reward' and 'penalty' associated with each client into our cost model. Specifically, we consider that every client can be assigned a reward indicating its value to the system (e.g. monetary value) when the client is successfully serviced, and conversely a penalty indicating the negative value (e.g. loss of profit) imparted to the system when the client is rejected in overloaded situations. These positive/negative parameters reflect the benefit/loss to the server system. Notice that only specifying the reward parameter without specifying the penalty parameter, or vice versa, is not sufficient. For example, ignoring penalties will lead to the naive design of reserving most or all of the system resources for low-priority clients in situations where low-priority clients arrive more frequently than high-priority clients, as there is no consequence in rejecting a high-priority client which may otherwise impose a very high penalty.

The motivation for introducing the reward/penalty parameters is to create a performance index which accounts for both the QoS and priority (importance) requirements of the client. It provides a basis for mathematically assessing the trade-off between priority reservation and no priority reservation designs. Under the priority reservation scheme, the server can reserve its capacity discriminatively for different priority-class clients for the purpose of 'reward optimization', considering that a high-priority user is associated with a higher 'value' if it is served successfully and, correspondingly, a higher 'penalty' if it is rejected. With this reward/penalty concept, the design of admission control algorithms can be considered as a reward optimization problem, i.e. designing an admission control algorithm which can dynamically adjust priority reservation policies as the input characteristics of client requests change dynamically, so as to maximize the system value. Dynamic workload change can happen during the peak/off-peak hours of an on-demand multimedia server system.

In this paper, we develop a class of 'threshold-based' admission control algorithms, with 'threshold' referring to the amount of server capacity reserved for different priority clients. We divide the server capacity into a number of 'priority threshold values' based on which the system can decide whether to accept high-priority or low-priority clients dynamically in order to maximize the system value. The resulting threshold-based admission control algorithm is developed based on the idea that admission control can be driven not only by hardware requirements, but also by knowledge of the workload characteristics of client requests, thus allowing the system dynamically to adjust the threshold values in response to changes in client workload characteristics. We derive a close-form expression for the value which the system can obtain when operating under the threshold-based algorithm as a function of model parameters, and discuss how the server can utilize the analytical solution at run time so as to maximize the system value dynamically. The contribution of our work with respect to previous works is that we vitalize the role of an admission control algorithm from a passive role, i.e. preventing system overload, to an active role, or maximizing system performance while still satisfying requests' continuity requirement. We achieve this goal by identifying the best priority reservation strategy when given a set of client workload characteristics. Our approach can handle multiple priority classes.

The rest of the paper is organized as follows. Section 2 presents the system model based on which several admission control algorithms are analysed, namely, free-threshold, fixed-threshold and dynamic-threshold admission policies. A performance criterion is defined which uses a combined reward/penalty performance metric for evaluating various threshold-based algorithms. Using simple queueing theory arguments, Section 3 derives closed-form expressions based on a performance metric in the form of $\mathcal{PO}_x(\bar{t}, \bar{s})$ where $\bar{t}$ is a set of priority threshold values and $\bar{s}$ is a set of input parameter values characterizing the behaviour of arriving clients in different priority classes. Section 4 discusses the optimality of the dynamic threshold-based algorithm, that is, being able to find an optimal $\bar{t}$ which maximizes $\mathcal{PO}_x(\bar{t}, \bar{s})$ for each $\bar{s}$ given. It also discusses how the analytical expressions derived in Section 3 can be used by the server

[1] Scheduling real-time transactions with values also concerns maximizing the system's added value; however, the main thrust is to design a transaction processing protocol by which the execution sequence of transaction operations can be ordered, possibly by aborting existing transactions or by delaying the commitment of transactions, so as to maximize the system value.

system at run time to adjust dynamically its threshold values so as to maximize the system value. Section 5 shows some numerical examples and gives physical interpretations of the result. Finally, Section 6 summarizes the paper and outlines some future research areas.

2. SYSTEM MODEL

We assume that the on-demand multimedia server adopts the *capacity reservation* mechanism such that a server capacity reservation is made at the time a new client arrives. A new client is accepted if the remaining capacity can accommodate it, otherwise, the client is turned away. Consequently, there exists a maximum number of client requests that the system can service without overloading, as has been addressed in previous works in admission control [8, 10]. Note that the above argument holds for both deterministic and best-effort admission control.

For ease of exposition, we first consider the case when there exist two priority classes of clients, each class being characterized by its own arrival/departure rates as well as its reward/penalty values. We will later relax this assumption in Section 3.

For trackability[2], we assume that the inter-arrival times of high-priority and low-priority clients are exponentially distributed with average times of $1/\lambda_h$ and $1/\lambda_l$, respectively. Once a reservation is made, a client is assured of the server capacity until its task is completed. The real-time computational requirement of each client is characterized by a period T and a computation time C within the period. To provide a real-time, continuous service to each client, a thread may be created by the server at the time the client is admitted into the system and is invoked afterward periodically, utilizing a fraction C/T of the server capacity, until the client completes its requested service. For simplicity, the inter-departure time of either the high-priority or low-priority clients is assumed to be exponentially distributed with an average time of $1/\mu$, although the analysis which follows can handle different departure rates.

The capacity reservation mechanism of the server system is modelled as follows. We assume that a high-priority client reserves a fraction $1/n$ of the capacity (corresponding to C/T above), whereas a low-priority client reserves a fraction $1/m$, $m \geq n$, of the capacity. In general, $m \geq n$ since a low-priority client may request a lower QoS requirement than a high-priority client. This paper considers the special case in which $m = n$, corresponding to the case where all clients' hardware requirement (e.g. playback rate of media streams) is the same but some clients are more important than others. 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 rejects a newly arriving client so as to guarantee continuous services to all clients that have been admitted. An example for which this assertion is justified occurs in the design of a real-time on-demand multimedia server [4] where the parameter n corresponds to the maximum number of subscriber requests with the same playback rate that can be serviced by the server. In such cases, as long as the system does not admit over n client requests, continuous services to all admitted client requests can be guaranteed.

The pay-off to the server when a client completes its service is characterized by each client's reward and penalty parameters. We assume that the rewards of high-priority and low-priority clients are v_h and v_l, respectively, with $v_h \geq v_l$, and 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 total pay-off rate, defined as the average amount of reward 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 and N_l low-priority clients per unit time while rejects M_h high-priority clients and M_l low-priority clients per unit time, then the system total pay-off rate is

$$N_h v_h + N_l v_l - M_h q_h - M_l q_l.$$

Formally, the problem we are thus interested in solving is to identify the best admission control policy under which this performance metric is maximized, as a function of model input variables, including n, λ_h, λ_l, μ, v_h, v_l, q_h and q_l defined above. Table 1 summarizes the notation used in this paper.

3. THRESHOLD-BASED ADMISSION CONTROL

In this section, we develop three threshold-based admission control algorithms in increasing order of reward maximization, at the expense of increased time complexity, and analyse their behaviours. We first derive analytical expressions for the system reward obtainable when only two priority classes exist. Later we will extend the analysis to the more general case when more than two priority classes exist.

3.1. Free threshold

The simplist admission policy, termed 'free-threshold', is to accept any new arriving client regardless of its priority type, as long as there is a slot to accommodate it. In essence, it is equivalent to the first come, first served policy without applying any reward-based control. We will use this policy as the base policy against which other more sophisticated admission policies can be compared. Essentially, under this policy, there is no priority distinction among clients and the system behaves like a classic $M/M/n/n$ system [14] with the arrival rate $\lambda = \lambda_h + \lambda_l$ and the departure rate $i\mu$ when there are i clients in the system. The loss rate of clients in

[2]Our approach can be applied to general distributions (e.g. a semi-Markov model) except that we will not be able to obtain analytical solutions and also a software tool which analyses stochastic models such as SHARPE [15] will have to be used to obtain numerical solutions on a case by case basis.

TABLE 1. Notation.

λ_h	Arrival rate of high-priority clients
λ_l	Arrival rate of low-priority clients
μ	Departure rate of clients
v_h	Reward of a high-priority client if the client is serviced successfully
v_l	Reward of a low-priority client if the client is serviced successfully
q_h	Penalty of a high-priority client if the client is rejected on admission
q_l	Penalty of a low-priority client if the client is rejected on admission
$\mathcal{PO}x$	Total system pay-off rate under admission policy x, e.g. income rate of the company running the on-demand multimedia service business
n	Maximum number of server capacity slots for servicing clients
n_h	Number of slots reserved for high-priority clients only, $0 \le n_h \le n$
n_l	Number of slots reserved for low-priority clients only, $0 \le n_l \le n$ and also $n_h + n_l \le n$
n_m	Number of slots that can be used to service either type of client, $n_m = n - n_h - n_l$
T	A time period in which a client's computation is executed periodically
C	The amount of computation time within T for each client; $C/T = 1/n$

this case is equal to

$$(\lambda_h + \lambda_l) \times \frac{\frac{1}{n!}\left(\frac{\lambda_h + \lambda_l}{\mu}\right)^n}{1 + \sum_{j=1}^{n} \frac{1}{j!}\left(\frac{\lambda_h + \lambda_l}{\mu}\right)^j},$$

where the first term is the collective arrival rate of high- and low-priority clients and the second term is the probability of all n slots being occupied.

Using only one component in the state representation, the $M/M/n/n$ model does not keep track of the number of high-priority or low-priority clients in each state. However, since with probability $\lambda_h/(\lambda_h + \lambda_l)$ a new client is a high-priority client and conversely with probability $\lambda_l/(\lambda_h + \lambda_l)$ it is a low-priority client, each state i (representing that there are i clients in the system in the steady state) can be associated with a pay-off reward rate of

$$i\mu \times \left(v_h \times \frac{\lambda_h}{\lambda_h + \lambda_l} + v_l \times \frac{\lambda_l}{\lambda_h + \lambda_l}\right).$$

On the other hand, state n (representing that all slots are used up) can be associated with a penalty rate of

$$q_h \times \lambda_h + q_l \times \lambda_l.$$

Consequently, the system pay-off rate under the free-threshold policy, $\mathcal{PO}_{free}$, can be obtained by summing the pay-off reward rates weighted by their individual state probabilities, and subtracting the penalty rates due to rejection when all n slots are occupied, i.e.

$$\begin{aligned}
&\mathcal{PO}_{free}(n, \lambda_h, \lambda_l, \mu, v_h, v_l, q_h, q_l) \\
&= \sum_{i=1}^{n} i\mu \times \left(v_h \times \frac{\lambda_h}{\lambda_h + \lambda_l} + v_l \times \frac{\lambda_l}{\lambda_h + \lambda_l}\right) \\
&\qquad \times \frac{\frac{1}{i!}\left(\frac{\lambda_h + \lambda_l}{\mu}\right)^i}{1 + \sum_{j=1}^{n} \frac{1}{j!}\left(\frac{\lambda_h + \lambda_l}{\mu}\right)^j} \\
&\quad - (q_h \times \lambda_h + q_l \times \lambda_l) \times \frac{\frac{1}{n!}\left(\frac{\lambda_h + \lambda_l}{\mu}\right)^n}{1 + \sum_{j=1}^{n} \frac{1}{j!}\left(\frac{\lambda_h + \lambda_l}{\mu}\right)^j}.
\end{aligned} \tag{1}$$

Note that here the total system pay-off rate $\mathcal{PO}_{free}$ is expressed as a function of n, λ_h, λ_l, μ, v_h, v_l, q_h and q_l.

3.2. Fixed threshold

Under the fixed-threshold admission policies, we allocate n_h slots, $n_h \le n$, to high-priority clients only, while the remaining slots $n_l = n - n_h$ slots are allocated to low-priority clients only. When all the slots allocated to high-priority (correspondingly low-priority) clients are exhausted, a new arriving high-priority (low-priority) client is rejected by the server even if there are still available slots in the slots allocated to low-priority (high-priority) clients. This situation is likely when we have a priori knowledge of the arrival rates of clients such that it is justified to reserve some capacity for high- or low- priority clients in order to maximize the system pay-off.

In this case, the server behaves as if it is managing two separate, concurrent queues: one is an $M/M/n_h/n_h$ queue for high-priority clients only with the arrival rate equal to λ_h, service rate equal to $i\mu$ when there are i high-priority

clients being serviced, and the number of slots equal to n_h, while the other is an $M/M/n_l/n_l$ queue designated for low-priority clients only with the arrival rate equal to λ_l, service rate equal to $i\mu$ when there are i low-priority clients being serviced, and the number of slots equal to n_l.

The loss rate of clients in this case is equal to the sum of that due to low-priority clients and that due to high-priority clients, i.e.

$$\lambda_h \times \frac{\frac{1}{n_h!}\left(\frac{\lambda_h}{\mu}\right)^{n_h}}{1+\sum_{j=1}^{n_h}\frac{1}{j!}\left(\frac{\lambda_h}{\mu}\right)^{j}} + \lambda_l \times \frac{\frac{1}{n_l!}\left(\frac{\lambda_l}{\mu}\right)^{n_l}}{1+\sum_{j=1}^{n_l}\frac{1}{j!}\left(\frac{\lambda_l}{\mu}\right)^{j}}.$$

By associating a pay-off reward rate of $i\mu \times v_h$ (correspondingly $i\mu \times v_l$) for state i in the $M/M/n_h/n_h$ queue for high-priority (correspondingly in the $M/M/n_l/n_l$ queue for low-priority) clients and subtracting the penalty rate for the case when n_h (correspondingly n_l) slots are used up for high-priority (low-priority) clients, we can compute the system pay-off rate as

$$\begin{aligned}
&\mathcal{PO}_{fixed}(n_h, n_l, \lambda_h, \lambda_l, \mu, v_h, v_l, q_h, q_l) \\
&= \sum_{i=1}^{n_h} i\mu \times v_h \times \frac{\frac{1}{i!}\left(\frac{\lambda_h}{\mu}\right)^{i}}{1+\sum_{j=1}^{n_h}\frac{1}{j!}\left(\frac{\lambda_h}{\mu}\right)^{j}} \\
&+ \sum_{i=1}^{n_l} i\mu \times v_l \times \frac{\frac{1}{i!}\left(\frac{\lambda_l}{\mu}\right)^{i}}{1+\sum_{j=1}^{n_l}\frac{1}{j!}\left(\frac{\lambda_l}{\mu}\right)^{j}} \\
&- \lambda_h q_h \times \frac{\frac{1}{n_h!}\left(\frac{\lambda_h}{\mu}\right)^{n_h}}{1+\sum_{j=1}^{n_h}\frac{1}{j!}\left(\frac{\lambda_h}{\mu}\right)^{j}} \\
&- \lambda_l q_l \times \frac{\frac{1}{n_l!}\left(\frac{\lambda_l}{\mu}\right)^{n_l}}{1+\sum_{j=1}^{n_l}\frac{1}{j!}\left(\frac{\lambda_l}{\mu}\right)^{j}}.
\end{aligned} \tag{2}$$

3.3. Dynamic threshold

Under the dynamic-threshold admission policy, the n slots are divided into three parts: n_h, n_l and n_m, with n_h specifically allocated to high-priority clients, n_l allocated to low-priority clients while the remaining n_m slots shareable with 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.

This policy encompasses the previous two admission policies: in the case when $n_m = 0$, this policy degenerates to the fixed-threshold admission policy, while in the case when $n_h = n_l = 0$, it degenerates to the free-threshold policy. It also covers the interesting case of $n_l = 0$ wherein high-priority clients can use the n_m slots open to both high- and low-priority clients (as long as there is a space), but no low-priority clients can use any of the n_h slots allocated to high-priority clients.

The closed-form solution to the system pay-off rate under the dynamic admission policy can be obtained by considering a two-level hierarchical model. The high-level model is like the one for the free-threshold admission policy, i.e. an $M/M/n_m/n_m$ queue with the arrival rate equal to $\Lambda_h + \Lambda_l$, and the service rate equal to μ. Here, Λ_h and Λ_l denote the arrival rates of high-priority and low-priority clients after n_h and n_l slots have been occupied by high-priority and low-priority clients, respectively. These two arrival rates associated with the 'spill-over' processes (which are Markov processes themselves [13]) can be obtained from two low-level models, one for each type of clients, like the ones we have used for the fixed-threshold admission policy. Specifically,

$$\Lambda_h = \lambda_h \times \frac{\frac{1}{n_h!}\left(\frac{\lambda_h}{\mu}\right)^{n_h}}{1+\sum_{j=1}^{n_h}\frac{1}{j!}\left(\frac{\lambda_h}{\mu}\right)^{j}}$$

and

$$\Lambda_l = \lambda_l \times \frac{\frac{1}{n_l!}\left(\frac{\lambda_l}{\mu}\right)^{n_l}}{1+\sum_{j=1}^{n_l}\frac{1}{j!}\left(\frac{\lambda_l}{\mu}\right)^{j}}.$$

The system pay-off rate in this case is the sum of that due to the n_h and n_l slots assigned to high- and low-priority clients only, and that due to the sharable n_m slots, i.e.

$$\begin{aligned}
&\mathcal{PO}_{dynamic}(n_h, n_m, n_l, \lambda_h, \lambda_l, \mu, v_h, v_l, q_h, q_l) \\
&= \sum_{i=1}^{n_h} i\mu \times v_h \times \frac{\frac{1}{i!}\left(\frac{\lambda_h}{\mu}\right)^{i}}{1+\sum_{j=1}^{n_h}\frac{1}{j!}\left(\frac{\lambda_h}{\mu}\right)^{j}} \\
&+ \sum_{i=1}^{n_l} i\mu \times v_l \times \frac{\frac{1}{i!}\left(\frac{\lambda_l}{\mu}\right)^{i}}{1+\sum_{j=1}^{n_l}\frac{1}{j!}\left(\frac{\lambda_l}{\mu}\right)^{j}} \\
&+ \mathcal{PO}_{free}(n_m, \Lambda_h, \Lambda_l, \mu, v_h, v_l, q_h, q_l),
\end{aligned} \tag{3}$$

where the expression for $\mathcal{PO}_{free}(\ldots)$ is given earlier in Equation 1. We check the boundary conditions below. For the special case when $n_h = n_l = 0$, we have $\Lambda_h = \lambda_h$, $\Lambda_l = \lambda_l$, the first two terms being zeros, and therefore $\mathcal{PO}_{dynamic}\ (0, n, 0,\ \ldots) = \mathcal{PO}_{free}(n,\ \ldots)$ in which the dynamic admission policy degenerates to the free admission policy. For the special case when $n_m = 0$, we have $\mathcal{PO}_{free}(0,\ \ldots) = -q_h\Lambda_h - q_l\Lambda_l$, and therefore $\mathcal{PO}_{dynamic}(n_h, 0, n_l,\ \ldots) = \mathcal{PO}_{fixed}(n_h, n_l,\ \ldots)$ in which the dynamic admission policy degenerates to the fixed admission policy. Equation 3 correctly satisfies these boundary conditions.

3.4. Multiple priority classes

Equations 1, 2 and 3 can easily be generalized to the case when there exist more than two priority classes. Assume that there are M priority classes, of which class 1 is the highest priority class, while class M is the lowest. Class k, $1 \leq k \leq M$, is characterized by its own set of parameters (λ_k, v_k, q_k). Other than the free algorithm, the threshold value specifically reserved for class k is n_k, with $\sum_{k=1}^{M} n_k = n$ for the fixed algorithm and $n_m + \sum_{k=1}^{M} n_k = n$ for the dynamic algorithm with n_m being the threshold value shareable to all priority classes. The generalization is straightforward and here we will only show the results without proof. Equations 4, 5 and 6 below give these reward rate expressions under the free, fixed and dynamical control algorithms, respectively, when M priority classes exist.

$$\mathcal{PO}^{M}_{free}(n, \lambda_1 \ldots \lambda_M, \mu, v_1 \ldots v_M, q_1 \ldots q_M)$$

$$= \sum_{i=1}^{n} i\mu \times \left(\sum_{k=1}^{M} \frac{v_k\lambda_k}{\lambda} \times \frac{\frac{1}{i!}\left(\frac{\lambda}{\mu}\right)^{i}}{1 + \sum_{j=1}^{n} \frac{1}{j!}\left(\frac{\lambda}{\mu}\right)^{j}} \right) \tag{4}$$

$$- \sum_{k=1}^{M} q_k\lambda_k \times \frac{\frac{1}{n!}\left(\frac{\lambda}{\mu}\right)^{n}}{1 + \sum_{j=1}^{n} \frac{1}{j!}\left(\frac{\lambda}{\mu}\right)^{j}}$$

where $\lambda = \sum_{k=1}^{M} \lambda_k$.

$$\mathcal{PO}^{M}_{fixed}(n_1 \ldots n_M, \lambda_1 \ldots \lambda_M, \mu, v_1 \ldots v_M, q_1 \ldots q_M)$$

$$= \sum_{k=1}^{M} \left(\sum_{i=1}^{n_k} i\mu \times v_k \times \frac{\frac{1}{i!}\left(\frac{\lambda_k}{\mu}\right)^{i}}{1 + \sum_{j=1}^{n_k} \frac{1}{j!}\left(\frac{\lambda_k}{\mu}\right)^{j}} \right) \tag{5}$$

$$- \sum_{k=1}^{M} \left(\lambda_k q_k \times \frac{\frac{1}{n_k!}\left(\frac{\lambda_k}{\mu}\right)^{n_k}}{1 + \sum_{j=1}^{n_k} \frac{1}{j!}\left(\frac{\lambda_k}{\mu}\right)^{j}} \right).$$

$$\mathcal{PO}^{M}_{dynamic}(n_1 \ldots n_M, n_m, \lambda_1 \ldots \lambda_M, \mu, v_1 \ldots v_M, q_1 \ldots q_M)$$

$$= \sum_{k=1}^{M} \left(\sum_{i=1}^{n_k} i\mu \times v_k \times \frac{\frac{1}{i!}\left(\frac{\lambda_k}{\mu}\right)^{i}}{1 + \sum_{j=1}^{n_k} \frac{1}{j!}\left(\frac{\lambda_k}{\mu}\right)^{j}} \right)$$

$$+ \ \mathcal{PO}^{M}_{free}(n_m, \Lambda_1 \ldots \Lambda_M, \mu, v_1 \ldots v_M, q_1 \ldots q_M) \tag{6}$$

where the expression for Λ_k was derived earlier in Section 3.3.

4. ALGORITHM COMPLEXITY AND SOLUTION TECHNIQUE

The inputs to the threshold-based admission control algorithms discussed above are the arrival and departure rates, i.e. λ_h, λ_l and μ, plus the reward and penalty parameters, i.e. v_h, v_l, q_h and q_l. The output is the overall system value, $\mathcal{PO}_{free}$ for free, $\mathcal{PO}_{fixed}$ for fixed and $\mathcal{PO}_{dynamic}$ for dynamic. In particular, for the last two outputs, we are interested in obtaining the system value obtainable at optimal conditions, that is, the largest $\mathcal{PO}_{fixed}$ optimal value (n_h, n_l) under the fixed algorithm, and the largest $\mathcal{PO}_{dynamic}$ optimal value (n_h, n_m, n_l) under the dynamic algorithm. Determining the optimal point in these cases can be done by enumerating all possible combinations and applying Equation 2 or Equation 3 to select the highest value. Specifically, the number of possible cases, which is the same as the number of ways of dividing n into K distinct groups (thresholds), is $C(n+K-1, K-1)$ with $C(x, y) = x!/(y!(x-y)!)$. Consequently, for the fixed and dynamic algorithms discussed in Subsections 3.2 and 3.3, the number of cases to be tested will be $C(n+1, 1) = n+1$ and $C(n+2, 2) = (n+2)(n+1)/2$, respectively. The time complexity involved in enumerating and applying Equation 2 or 3 is thus $O(n)$ for fixed and is $O(n^2)$ for dynamic.

It is easy to show that under a given set of parameter values, there always exists an optimal threshold value set

FIGURE 1. A 3-D graph showing (n_h, n_m, n_l) vs. $\mathcal{PO}_{dynamic}$. The maximum point is (7, 13, 0).

of (n_h, n_m, n_l) under which the system value is maximized under the dynamic algorithm. The reason is twofold. First, the number of cases which can be enumerated for a given n under the dynamic algorithm is finite, i.e. $(n+2)(n+1)/2$ to be exact. Therefore, by using Equation 3 to compute the system value obtained for each case, we can determine the optimal set which yields the maximum system value. Second, all the cases enumerated by either the free or fixed algorithm are just subcases which can be enumerated under the dynamic algorithm. The free algorithm generates one special case for which $n_h = n_l = 0$ and $n_m = n$, while the fixed algorithm generates $n+1$ special cases for which $n_m = 0$. Therefore, the optimal threshold value set will always be uncovered by the dynamic algorithm.

There are two ways of applying the analysis result obtained in this paper to real-time admission control. The first way is to statically generate a table and then do a table lookup at run time. This method is applicable when v_h, v_l, q_h and q_l can statically be determined by the system designer at the design time based on the characteristics of client service classes and the application environment. In this case, we can evaluate optimal (n_h, n_m, n_l) sets statically for various combinations of λ_h, λ_l and μ which are likely to change dynamically. A symbolic mathematical software package such as Mathematica [16] can be used for this purpose. Figure 1 shows a three-dimensional graph generated by Mathematica. It shows $\mathcal{PO}_{dynamic}$ as a function of (n_h, n_m, n_l) for the case when $n = 20, v_h = 10, v_l = 2, q_h = 2, q_l = 1$ and $\lambda_h = 5, \lambda_l = 40$ and $\mu = 1$. Note that n_m is not shown on the graph because it is equal to $n - n_h - n_l$. The optimal set as determined by Mathematica in this case is (7, 13, 0). After the table is statically established this way to cover a range of client arrival and departure rates, the system can (a) collect run-time client arrival and departure data periodically; (b) estimate the average arrival and departure rates in the period; and (c) adjust the optimal threshold value set by using a look-up table so as to optimize the system pay-off value dynamically on a period by period basis. This method is feasible for video server designs since it is reasonable to assign clients in different priority classes with distinct reward/penalty values at the design time based on the belief of the designer.

The second way is to treat v_h, v_l, q_h and q_l also as variables and apply enumeration methods and Equation 3 at run-time to select the optimal threshold values. Since the algorithm's complexity is $O(n^2)$, it should be done only periodically, perhaps by executing a background process which periodically estimates average parameter values and

recomputes the next set of optimal (n_h, n_m, n_l) values to be used by the server in the next period. The server can continue using the old set of (n_h, n_m, n_l) values to admit users while the background process computes the new optimal set.

5. NUMERICAL EXAMPLES

We use the design of an on-demand multimedia server [10] as an example to demonstrate the utility of our analysis result. One reported experiment used an array of 128 disks each with a storage capacity of 0.5 Gbyte to implement the server. The playback rate is assumed to be 30 frames/s per client request. Successive media blocks (each of 512 kbyte) of a video stream are assumed to be randomly stored on disk. Under the hardware constraints of the disk array used the maximum number of client requests that can be served concurrently was found to be around $n = 16$ if deterministic admission control is considered and $n = 84$ if best-effort admission control is considered with a read-ahead buffer of 1 media block per client. The latter is different from the former in that the server admits clients based on the observed performance characteristics of the server with predictive guarantee, rather than based on theoretical worst-case time bounds with absolute guarantee [10]. It should be noted that this analysis was purely based on resource capacity limitations, without considering the importance or criticality of requests.

Tables 2 and 3 list the optimal (n_h, n_m, n_l) threshold value sets with respect to some selected sets of model parameter values characterizing various client workload possibilities for the server system, for $n = 16$ and $n = 84$, respectively. Tables 2 and 3 are generated by applying Equations 1 and 3. In addition, the values listed in the column labelled 'optimal (n_h, n_m, n_l)' are uncovered by the dynamic admission control algorithm based on the mechanism discussed in Section 4.

Here we observe that the pay-off rate at the optimizing condition under dynamic-threshold admission can be much higher than that under free-threshold admission. Moreover, the total pay-off rate can be negative if requests are served indiscriminatively. Another observation is that as the arrival rate (λ_h), reward (v_h) or penalty (q_h) of high-priority clients increases, more slots will be reserved for high-priority clients. An example is entry 1 of Table 2 for which the number of slots reserved for high-priority clients is zero and therefore a high-priority client (who still presumably pays more) will have to compete with low-priority clients for the shareable slots. The reason is that the arrival rate of high-priority clients in entry 1 is an order of magnitude lower than that of low-priority clients (i.e. 1 versus 10) and also the reward of accepting a high-priority client is not high compared to that of accepting a low-priority client (i.e. 2 versus 1). The last entry of Table 2 shows that as both the arrival rate and reward of high-priority clients increase (relative to those of low-priority clients), more and more slots will be reserved for high-priority clients by the system in order to maximize the total system pay-off.

Figure 2 demonstrates the effect of applying dynamic threshold admission control more clearly. It displays the difference between the optimal pay-off rate and that without control (i.e. with free-threshold), with λ_h varying in the range of [10, 150] in increments of 10 and λ_l varying in the range of [50, 150] in increments of 50, and with $n = 100$. We set the reference reward/penalty parameter values for low-priority clients at 1 and let $v_h/v_l > q_h/q_l$, meaning that the loss the system suffers from rejecting a high-priority client is lower in absolute value than the reward that it receives from accepting the same high-priority client. We study this case because it is generally true that the system would not lose a value due to turning away a client more than it would gain due to accepting the same client, especially for high-priority clients. The trend exhibited in Figure 2 can be explained as follows. When the system load is light (corresponding to the left part of the graph) the effect of threshold-based admission control is not significant because the system can accommodate a new arriving client with a high probability, so that free admission is just as good as threshold-based admission. As the system load becomes moderate to heavy (corresponding to the middle part of the graph) the effect of threshold admission control becomes manifested because the server can effectively manage the server capacity (with thresholds) based on the knowledge regarding workload characteristics so as to optimize the pay-off rate. Finally, when the load is very heavy (corresponding to the right part of the graph) the effect of threshold admission control becomes less significant again because too many clients are lost even if the dynamic control algorithm

TABLE 2. Optimizing threshold values ($n = 16$).

$(\lambda_h, \lambda_l, \mu, v_h, v_l, q_h, q_l)$	optimal (n_h, n_m, n_l)	dynamic $\mathcal{PO}$	free $\mathcal{PO}$
(1, 10, 1, 2, 1, 2, 1)	(0,7,9)	12	11
(1, 10, 1, 5, 1, 2, 1)	(1,7,8)	15	14
(1, 10, 1, 10, 1, 2, 1)	(1,7,8)	19	18
(5, 10, 1, 2, 1, 2, 1)	(3,10,3)	15	14
(5, 10, 1, 5, 1, 2, 1)	(5,10,1)	28	27
(5, 10, 1, 10, 1, 2, 1)	(6,10,0)	52	48
(10, 10, 1, 2, 1, 2, 1)	(8,8,0)	14	12
(10, 10, 1, 5, 1, 2, 1)	(12,4,0)	41	33
(10, 10, 1, 10, 1, 2, 1)	(14,2,0)	88	69

TABLE 3. Optimizing threshold values ($n = 84$).

$(\lambda_h, \lambda_l, \mu, v_h, v_l, q_h, q_l)$	optimal (n_h, n_m, n_l)	dynamic $\mathcal{PO}$	free $\mathcal{PO}$
(10, 100, 1, 2, 1, 2, 1)	(7,60,17)	59	57
(10, 100, 1, 5, 1, 2, 1)	(10,68,6)	86	79
(10, 100, 1, 10, 1, 2, 1)	(12,72,0)	134	117
(50, 100, 1, 2, 1, 2, 1)	(50,34,0)	49	20
(50, 100, 1, 5, 1, 2, 1)	(56,28,0)	192	103
(50, 100, 1, 10, 1, 2, 1)	(60,24,0)	436	242
(100, 100, 1, 2, 1, 2, 1)	(84,0,0)	22	−50
(100, 100, 1, 5, 1, 2, 1)	(84,0,0)	263	75
(100, 100, 1, 10, 1, 2, 1)	(84,0,0)	666	283

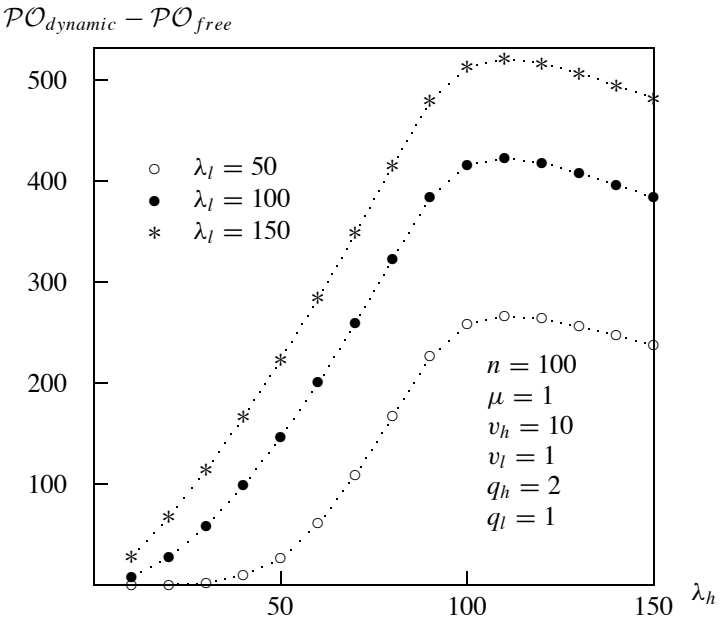

FIGURE 2. Difference in reward rate as a result of applying threshold-based reward-optimization admission control.

is in effect.

It should be noted that the trend exhibited in Figure 2 is not universally true for all cases. It depends on the relative ratios of v_h/v_l and q_h/q_l and, in general, the characteristics of the workload for the server system in question. The equations derived in the paper allow the designer to identify the optimizing (n_h, n_m, n_l) set under a specified workload condition, and quantitatively predict how much benefit the system can gain (e.g. in terms of reward rate) by employing the threshold-based admission control algorithm.

6. CONCLUSIONS

In this paper, we have analysed a design concept for implementing reward-optimization admission control algorithms for on-demand multimedia systems. The design concept is based on the idea that an admission control program should consider not only the underlying hardware limitation of the system, but also the benefit it can bring to the system as a result of accepting/rejecting client requests. We illustrated our concept by using the system's reward rate (e.g. the income rate of a company that runs the on-demand multimedia server business) as a metric to guide the design of admission control algorithms. We investigated a class of threshold-based admission control algorithms for maximizing this metric. Analytical expressions for the system pay-off rate under these threshold-based admission control algorithms were derived. They can be used to determine the optimal condition for accepting/rejecting client requests and help the system designer determine in a quantitative way how much benefit the system will gain as a result of applying a threshold-based admission control algorithm. Finally, the effectiveness of our approach was demonstrated by a realistic on-demand multimedia server. Our approach is particularly useful for situations where the system's client-priority/workload characteristics may change dynamically during its peak/off-peak hours. The technique presented in the paper can be used to adjust dynamically the threshold values based on the system characteristics so that the system can always receive the best reward without violating its continuity requirement.

Some future research areas include (a) coupling the concept of reward-optimization with lowering the quality of service (QoS) levels of existing clients so as to make room for new arriving clients; and (b) designing threshold-based admission control algorithms for on-demand multimedia systems in which clients may have different rewards or penalties when successfully or unsuccessfully served as well as different QoS requirements.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Council of R.O.C. under grant NSC-86-2745-E-006-020.

REFERENCES

[1] Mei, G. G., Lin, M. H., Hu, L. and Chang, H. (1992) A real-time multimedia system for video applications. *26th IEEE Conf. Signals, Systems and Computers*, Pacific Grove, CA, pp. 1031–1036.

[2] Oomoto, E. and Tanaka, K. (1993) OVID: Design and implementation of a video-object database system. *IEEE Trans. Know. Data Engng*, **5**, 629–643.

[3] Oyang, Y. J., Wen, C. H., Cheng, C. Y., Lee, M. H. and Li, J. T. (1995) A multimedia storage system for on-demand playback. *IEEE Trans. Consumer Electronics*, **41**, 53–64.

[4] Rangan, P. V., Vin, H. M. and Ramanathan, S. (1992) Designing an on-demand multimedia service. *IEEE Commun.*, **30**, 56–64.

[5] Vina, A., Lerida, J. L., Molano, A., and del Val, D. (1994) Real-time multimedia systems. *13th IEEE Symp. Mass*

Storage Systems,, Annecy, France, pp. 77–83.

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

[7] Fujikawa, K. *et al*. (1995) Application level QoS modeling for a distributed multimedia system. *1995 Pacific Workshop Dist. Multimedia Systems*, Honolulu, Hawaii, pp. 44–51.

[8] Chang, E. and Zakhor, A. (1996) Cost analysis for VBR video servers. *IEEE Multimedia*, **3**, 56–71.

[9] Ramanathan, S. and Rangan, P. V. (1994) Architecture for personalized multimedia. *IEEE Multimedia*, **1**, 37–46.

[10] Vin, H. M., Goyal, A. and Goyal, P. (1995) Algorithms for designing multimedia servers. *Computer Commun.*, **18**, 192–203.

[11] Bestavros, A. and Braoudakis, S. (1995) Value-cognizant speculative concurrency control. *VLDB'95: The International Conf. on Very Large Databases*, Zurich, Switzerland, September, pp. 122–133.

[12] Locke, C. (1986) *Best Effort Decision Making for Real-Time Scheduling*. Ph.D. Thesis, Carnegie-Mellon University, Department of Computer Science, Pitterburg, PA.

[13] Sahner, R. A., Trivedi, K. S. and Puliafito, A. (1996) *Performance and Reliability Analysis of Computer Systems*. Kluwer Academic Publishers, Boston, MA, pp. 71–72.

[14] Kleinrock, L. (1975) *Queueing Systems, Vol. 1: Theory*. John Wiley and Sons, New York.

[15] Sahner, R. A. and Trivedi, K. S. (1991) *SHARPE Language Description*. Duke University, Durham, NC.

[16] Wolfram, S. (1996) *Mathematica 3.0*. Cambridge University Press, Cambridge, UK.