Multimedia Tools and Applications, 25, 167–185, 2005
© 2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Adaptive QoS Control Based on Benefit Optimization for Video Servers Providing Differentiated Services

ING-RAY CHEN* irchen@cs.vt.edu
Department of Computer Science, Virginia Tech, Northern Virginia Graduate Center, 7054 Haycock Road, Falls Church, VA 22043, USA

SHENG-TUN LI stli@mail.ncku.edu.tw
Institute of Information Management, National Cheng Kung University, 1 University Road, Tainan, Taiwan, Republic of China

I-LING YEN ilyen@utdallas.edu
Department of Computer Science, University of Texas at Dallas, Arlington, Texas, USA

Abstract. We propose and analyze quality of service (QoS) control algorithms for video servers designed to provide differentiated video streaming services. The design concepts are based on resource reservation and *benefit optimization* so that resources are reserved dynamically and adaptively for different QoS levels in response to the changing workload of the system, with the objective of maximizing the *benefit throughput* obtainable by the system. We analyze the benefit throughput obtainable by the system for a baseline algorithm for which the QoS levels of admitted users are not changed during the service lifetime and a greedy algorithm that may raise QoS levels of admitted users due to resources being free from departure events. We validate the design of these two QoS control algorithms via a detailed simulation study.

Keywords: video servers, quality of service (QoS), QoS negotiation, streaming video, admission control, disk scheduling, data placement

1. Introduction

With the deployment of broadband technologies and services lately, it is anticipated that video streaming over the Internet will soon become a reality within a few years [8]. To deal with the exponential growth of users demanding streaming services, it is critical to use efficient quality of service (QoS) control schemes at both the server side and network switches to fully exploit multiplexing benefits in bandwidth management. Over the past few years, there has been substantial research effort in the area of QoS control schemes working toward this goal. On the server end, various algorithms have been proposed to optimize disk bandwidth allocations for effectively servicing multiple requests by means of

*This work of Dr. Ing-Ray Chen is supported in part by a National Science Foundation grant #9987586, a Microsoft Research grant and an Intel grant.

admission control and data placement schemes [5]. The goal is either to reduce the overall cost per request or to maximize the maximum number of requests the system is able to service simultaneously with statistical or absolute QoS guarantees. There also have been research efforts to deal with mixed workloads, again by means of admission control and disk scheduling mechanisms [13, 14].

This paper also concerns with the server end design. In particular, it addresses the QoS control and negotiation issues for a video server that provides streaming services to concurrent requests. Here, by "QoS" control, we refer to controlling disk/memory resources at the server end to satisfy the bandwidth requirements of user video streams. Some of these QoS control issues considered in the paper have been partly addressed in other multimedia server designs [2–4, 6, 9–11]. Lee and Sabata [9] propose the concept of *benefit functions* and *resource demand functions* associated with application requests, characterizing each application with a dimension (or a range) of QoS requirements and its corresponding "benefit" values with which the application brings to the system. They use the concept of *benefit optimization* to design QoS-based admission control algorithms, given that each application is characterized by its own specific set of benefit and resource demand functions. A key design of their work is that a portion of the resources is reserved specifically to serve requests with degraded QoS. The design thus performs QoS negotiation and adjustment only to requests admitted into the reserved area. When a request departs, their algorithm adjusts the QoS levels of requests admitted into the reserved area with the goal of maximizing the benefit. A shortcoming of their work is that it does not discuss how much resources should be reserved dynamically in response to changing workloads so that the system benefit is maximized at run time. It also requires each application to supply a (continuous) benefit function based on QoS levels (or resources) received from the system; such information requires the application to know a priori how many levels of QoS are offered by a multimedia server.

Cheng et al. [3, 4] in their work consider a class of priority-based admission control algorithms for optimizing the same benefit value. They consider a simpler form of benefit functions, i.e., each application is associated with a benefit value if it is served with a particular QoS level with the number of QoS levels defined by the system. In addition, they introduce the concept of "penalty" functions to characterize the loss to the system when requests must be rejected by the system due to admission control. High-priority requests are always served with the highest QoS level, while low-priority requests can have a range of QoS levels, each characterized by a benefit value. They divide the resources into three parts: one for serving the high-priority requests with the highest QoS level, one for serving the low-priority requests also with the highest QoS level, and lastly a reserved area for serving both high- and low-priority with QoS adjustment being performed only on low-priority requests. They identify the best partition of these three areas for allocating resources dynamically so as to optimize the system benefit at run time. Their analysis, however, is based on hypothetical QoS levels (two) with no real data support.

In this paper, we develop a new periodic greedy reservation algorithm based on the use of benefit/penalty functions, with the goal of optimizing a benefit-based performance metric of the system, defined as the benefit throughput, or amount of benefit values earned per time unit. Our work contributes to the field of QoS management and admission control as follows.

First, we consider true QoS characteristics of a video server via actual trace data. Thus, the QoS levels and their associated resource consumptions are not hypothetical. Second, we give a theoretical derivation of the benefit throughput obtainable from our algorithm developed, when given the benefit/penalty and input traffic information of requests. Third, our algorithm is executed at run time utilizing the theoretical results obtained at static time for adjusting the resource allocation greedly and periodically so as to optimize the performance metric dynamically. Lastly, we develop a full disk simulation model for a video server to evaluate the merit of our periodic greedy reservation algorithm compared with other QoS control schemes.

The rest of the paper is organized as follows. Section 2 gives the system model and states the system assumptions. Section 3 first derives the theoretical benefit throughput obtainable by considering a baseline QoS control scheme. Then it describes a periodic, greedy reservation algorithm by utilizing the theoretical results as a basis to further optimize the benefit throughput at run time. The complexity of these QoS control algorithms based on benefit optimization is then analyzed, along with a discussion of how the concept of benefit optimization can be applied to video servers designed to provide streaming services to users in distinct priority classes. Section 4 presents a simulation model for validating the baseline and greedy QoS control algorithms developed in the paper. Physical interpretations of the simulation results are given. Finally, Section 5 concludes the paper and outlines some future research areas.

2. Background and system model

We assume the video server stores on a single disk multiple video titles, each of which has multiple QoS versions. For example, a video title may have three versions stored on disk. The first version is of the highest QoS level, e.g., a compressed MPEG file. The second version is of the medium QoS level, e.g., a compressed H-263 video file at 64 Kbps $\times$ p, where $1 \leq p \leq 32$. The final version is of the lowest QoS level, say, based on H-261 format at 64 Kbps. For scalable videos such as MPEG4, it is also possible to apply a scalable codec at static time and store the original version along with several scaled-down versions of the same video title on disk, with each demanding a different disk bandwidth requirement. Note that storing a single copy of a scalable MPEG-4 video on disk and then applying a scalable codec to produce scaled videos at run time would not reduce the disk bandwidth requirement since we need to first retrieve the MPEG-4 video from disk before decoding can be performed.

We assume that users request streaming services from the video server with an arrival rate of λ which may change from time to time and a departure rate of μ. No priority classes exist for users.[1] The video server provides streaming services to admitted users concurrently in cycles so it can use efficient disk scheduling algorithms [5, 12] to minimize the disk seek overhead when retrieving data needed by all users in each cycle. Assume that the cycle time is T_{SR}, e.g., 1 second. Further, the video server executes an admission control algorithm to control who can be admitted into the system and also what QoS level should be given to an admitted user request so that the total time used to retrieve data needed by all admitted users in any cycle does not exceed T_{SR} (deterministically or statistically). When a user request departs, the video server executes a resource reclamation algorithm to deallocate the resources allocated to the departing user, or distribute the resources to other user requests. Once the system admits a user, the user remains in the system until the service is completed. We assume that it is disturbing to the user to arbitrarily raise and lower the QoS level during the service lifetime since users may prefer to stay at a particular QoS level upon admission till it departs. Our baseline QoS control algorithm follows this assumption. While our model eliminates the scenario of arbitrary QoS changes, we assume that in some systems users can accept their QoS levels being raised (but never lowered) during the service lifetime. Our greedy QoS control algorithm follows this assumption.

We assume that all video titles stored in the multimedia servers have the same number of QoS levels. Further, we assume that each QoS level i is associated with a benefit value v_i, with $v_j < v_i$ if $j > i$. That is, the highest QoS version has v_1, and the second highest QoS version has v_2, and so on. The assignment of benefit values to QoS levels is done by the service provider (e.g., pay per view) and the benefit each admitted user request brings to the system depends on the QoS received by the user during its service period. For systems that the QoS level granted to a user is never changed during the service lifetime, e.g., fixed at level i as determined upon admission, the user will pay v_i when it departs. Our baseline QoS control algorithm follows this assumption. For systems that allow the QoS level to be raised during the service lifetime, we assume that the benefit value obtainable by the system from a departing user is proportional to the service periods of QoS levels. Suppose the total service time of a user request is T before it departs. Also suppose that during one third of the service time the user receives only the 2nd-level QoS (with benefit v_2) and during the remaining time, it receives the highest QoS level (with benefit v_1). Then the total benefit that this user brings to the system when it departs is $v_2/3 + 2v_1/3$. For such systems, the user expects a possible (although not guaranteed) QoS improvement but never a QoS degradation. The users of the system are aware of differentiated QoS services provided by the server and that the price to pay depends on the QoS service received.

As more users are in the system consuming all the system resources, the system has to reject new users. We assume the system will not lower the QoS of admitted users to make room to accommodate new users. To model the effect of rejection, we adopt the notion of a penalty value q being removed from the benefit when the system rejects a user. Adding this penalty parameter to the cost model is essential because otherwise we can always admit users into the highest QoS level only until resources are exhausted and then reject all new arrivals after that so that the benefit received is maximized. From the service provider's viewpoint, there is always some loss incurred to the system when a user is rejected (e.g., unhappy users); we assume that the penalty parameter is assessed by the service provider. It can be set to zero if rejecting users is not an issue in the application.

3. Reservation algorithms for QoS control

The QoS control algorithms developed in this paper is based on resource reservation. We partition the system resources into multiple parts, one for each QoS level. For example, 80% of resources are for the highest QoS; 15% are for the medium QoS and the remaining 5% are for the lowest QoS if three QoS levels exist. The system always admits the users into the highest QoS region whenever possible; if resources over there are all used-up, it will admit users into the next QoS region, and so on.[2] The best partition of these multiple QoS admission regions for maximizing the benefit is dictated by the user arrival/departure rate and benefit/penalty functions. For example, when the arrival rate is low (so the rejection probability is low) or when v_1 is high, it can be that we will allocate 100% to the highest QoS only, thus leaving no room for other QoS levels.

We first derive an analytical expression for the theoretical benefit throughput obtainable from a baseline QoS control algorithm. Recall that again the baseline QoS control algorithm is based on the assumption that once a user is admitted, it will stay at the admitted QoS level till it departs. We later will apply the analytical results to guide the design of the greedy algorithm at run time, with the goal to further maximize the benefit throughput dynamically. Recall also that the greedy algorithm is based on the assumption that the the system can raise (but not lower) the QoS level of users during the service lifetime.

3.1. Baseline algorithm

Without loss of generality, assume that there are M QoS levels with benefit values v_i, $1 \leq i \leq M$, respectively. For a video server, it is possible for us to know a priori the resource (bit rate) requirements of streaming requests at these M QoS levels at the static time via trace data. Thus, if we are given a resource partition, say, $(B_1, B_2, \ldots, B_M)$, where B_i stands for the disk bandwidth allocated to QoS level i and $\sum_{i=1}^{M} B_i = B$, the total disk bandwidth, then based on statistical admission [5] we can estimate the maximum number of users in each QoS level that the system is able to accept with a threshold disk overload probability, i.e., 10^{-4}. Let the set of maximum number of user requests statistically admissible be $(n_1, n_2, \ldots, n_M)$ where n_i stands for the maximum number of users admissible in QoS i level given that the bandwidth allocation is B_i.

Consider a baseline QoS control scheme as follows. The system admits up to a maximum of n_1 users in the highest QoS level; then it admits users up to a maximum of n_2 users in the next QoS level and so on. When a user of QoS level i departs, no QoS adjustment is made to users in QoS level j, $j > i$, i.e., no QoS enhancements, so once a user is admitted into QoS level i, it stays in that level until the service is completed. When slots are free due to departures, the vacant slots are filled with new arrivals only; the system always fills slots in decreasing order of QoS. Thus, the system behaves as if it contains M queues with sizes n_i's, $1 \leq i \leq M$, respectively, one for each QoS level. In other words, the system behaves as if it contains M $/m/m/n_i/n_i$ queues [7], $1 \leq i \leq M$, each having n_i QoS slots to hold users at QoS level i. Each of these M queues is characterized by its input user arrival rate and its per-user departure rate. The per-user departure rate is the same for all queues, i.e., μ. The input arrival rate to the queue of the highest QoS level is λ itself. The input arrival rate to the queue of QoS level $i + 1$ is the *spill-over* rate from the queue of QoS level i when the latter queue is full. Specifically, the input arrival rate to queue 1 is $\lambda_1 = \lambda$, and the input arrival rate to queue $i + 1$, λ_{i+1} for $i > 1$, is equal to the input arrival rate to the queue of QoS level i (i.e., λ_i) multiplied with the probability that all n_i slots at QoS level i are filled, i.e.,

$$\lambda_{i+1} = \lambda_i \times \frac{\frac{1}{n_i!}\left(\frac{\lambda_i}{\mu}\right)^{n_i}}{1+\sum_{j=1}^{n_i}\frac{1}{j!}\left(\frac{\lambda_i}{\mu}\right)^j}$$

The system rejects incoming users only when all its resources are used up, i.e., when all slots are used up in all M QoS levels. Thus, the rejection rate is equal to the input arrival rate to the queue of the lowest QoS level (i.e., λ_M) multiplied by the probability that all n_M slots at QoS level M are filled, i.e.,

$$\lambda_{rej} = \lambda_M \times \frac{\frac{1}{n_M!}\left(\frac{\lambda_M}{\mu}\right)^{n_M}}{1+\sum_{j=1}^{n_M}\frac{1}{j!}\left(\frac{\lambda_M}{\mu}\right)^j}$$

When a user is rejected, the system loses a profit of q and when a user at QoS level i departs, the system earns a benefit of v_i. Therefore, under this baseline QoS control scheme, the resulting benefit throughput is the sum of that due to users departing at all the M QoS levels, minus the penalty rate due to users being rejected when all resources are used-up in all QoS levels. Let $\mathcal{BT}$ be the benefit throughput obtainable under the baseline QoS control scheme. Then,

$$\begin{aligned}\mathcal{BT} = {} & \sum_{i=1}^{M}\sum_{k=1}^{n_i} k\mu \times v_i \times \frac{\frac{1}{k!}\left(\frac{\lambda_i}{\mu}\right)^{k}}{1+\sum_{j=1}^{n_i}\frac{1}{j!}\left(\frac{\lambda_i}{\mu}\right)^j} \\ & - q \times \lambda_M \times \frac{\frac{1}{n_M!}\left(\frac{\lambda_M}{\mu}\right)^{n_M}}{1+\sum_{j=1}^{n}\frac{1}{j!}\left(\frac{\lambda_M}{\mu}\right)^j}\end{aligned} \tag{1}$$

where $\lambda_1 = \lambda$ and λ_{i+1} is related to λ_i, $1 \le i < M$, as given earlier. The last term accounts for the penalty rate due to rejecting users.

When given parameter values of $\lambda, \mu, v_1, v_2, \ldots, v_M$, and q, one can statically determine the best partition of $(n_1, n_2, \ldots, n_M)$, say, $(n_1^*, n_2^*, \ldots, n_M^*)$, for maximizing the benefit throughput. To search for the best partition set, we formulate the search problem as a linear programming problem as follows. Let N_i be the maximum number of requests at QoS level i that the system is able to admit statistically (with a threshold disk overload probability of 10^{-4}) when all the cycle duration period T_{SR} is allocated to service requests at QoS level i only. Thus, there will be M parameters, i.e., $N_1, N_2, \ldots, N_M$, for M QoS levels, respectively, which we can compute at static time [5]. Then the best partition set $(n_1^*, n_2^*, \ldots, n_M^*)$ is the one that maximizes $\mathcal{BT}$ (using the equation derived above) subject to the condition that:

$$\sum_{i=1}^{M-1}\left(\left\lfloor\frac{N_M}{N_i}\right\rfloor \times n_i\right) + n_M = N_M. \tag{2}$$

3.2. Greedy QoS-control algorithm

For systems that allow the QoS level to be increased during the service lifetime of users, we use a greedy algorithm to further improve the benefit throughput obtainable. In fact, the resulting benefit throughput obtainable from the baseline QoS control scheme will become a lower bound for the greedy algorithm.

We first notice that the baseline QoS control scheme does not allow resources to be redistributed when a user departs. That is, the system always statically admits users up to a maximum of n_1 at the highest QoS level and when all n_1 slots are filled up, it then admits users at the next inferior QoS level up to a maximum of n_2 and so on. Hence, if the system is in a state in which n_1 slots are all filled but n_2 slots are just partially filled, then if later on a user using one of the n_1 slots departs, the system would not do anything with the vacant slot in n_1; instead, it just waits until another user arrives. This is a waste of valuable resources especially if vacant slots due to user departures are of the highest QoS quality.

We modify our algorithm to be greedy as follows: When a user in QoS level i departs, if a user exists at the next inferior QoS level $i + 1$, then that user's QoS is raised to QoS i; otherwise, raise the QoS of a user at QoS $i + 2$ to QoS i, if a user is found at QoS level $i + 2$, and so on. Note that since the user departure time is independently distributed, it is possible that a user admitted into level $i + 1$ can depart earlier than a user admitted into level $i + 2$ or even a lower QoS level. Also, when a user at QoS level $i + 1$ is promoted to QoS level i, creating an empty slot at level $i + 1$, we repeat the same process to raise the QoS level of a user at $i + 2$ to $i + 1$, and if a user is not found at level $i + 2$, we go to level $i + 3$, and so on. Thus a departure of a user at level i triggers a ripple-promotion effect and potentially $M - i$ users, one from each of the following $M - i$ levels, can have their QoS levels promoted by one level. As for the selection of the user for QoS upgrade, conceivably there are several selection policies available such as random, shortest-time first, longest-time first, etc. For fairness, this paper adopts the longest-time first policy such that a user that stays at level i for the longest time among all users in the same level will be selected for QoS upgrade. Notice this greedy policy always improves the benefit throughput. Also, it does not disturb users since the greedy algorithm only raises the QoS, which in general is likely to be acceptable to users.

3.3. Periodic and dynamic resource reservation

In both the baseline and greedy algorithms, we use a look-up table listing the best $(n_1^*, n_2^*, \ldots, n_M^*)$ value set for each arrival rate value, along with the benefit throughput obtainable at that arrival rate. This table can be obtained at the design time from the analysis given in Section 3.1. Our algorithms are dynamic and periodic in resource reservations such that the best $(n_1^*, n_2^*, \ldots, n_M^*)$ value set is changed dynamically in response to workload changes at run time. The workload change can be profiled based on historical data collected identifying arrival rates during peak and off-peak time periods during a day. It can also be monitored by observing the number of users arriving at the system over an observation period such that the arrival rate is estimated by dividing the number of users by the observation time period.

Table 1. Baseline (n_1^*, n_2^*, n_3^*) and benefit throughput.

λ (arrivals/minute)	(n_1^*, n_2^*, n_3^*)	Benefit throughput (sec^{-1})
10	(41, 69, 4)	2.33
15	(28, 133, 6)	2.94
20	(15, 197, 8)	3.54
25	(10, 197, 58)	3.94
30	(4, 197, 118)	4.22
35	(0, 197, 158)	4.48
40	(0, 174, 204)	4.49
45	(0, 129, 294)	4.48
50	(0, 83, 386)	4.46
55	(0, 36, 480)	4.45
60	(0, 0, 552)	4.44

$v_1 = 20$, $v_2 = 10$, $v_3 = 5$ and $q = 1$.

Based on the arrival rate monitored at run time or profiled a priori, the system then performs a table lookup operation at run time to determine the best $(n_1^*, n_2^*, \ldots, n_M^*)$ value set based on the estimated current arrival rate to maximize the benefit throughput obtainable. Here we again emphasize that the calculation of the best $(n_1^*, n_2^*, \ldots, n_M^*)$ value set is done at static time and only a table look-up operation is required to be performed at run time in response to changing workloads. Table 1 lists an example table for $M = 3$ to be used at run time. During an off-peak period in which the arrival rate is 10 arrivals/minute, then $(n_1^*, n_2^*, n_3^*) = (41, 69, 4)$ should be used. If the arrival rate monitored is changed to 30 arrivals/minutes during a peak period, on the other hand, then $(n_1^*, n_2^*, n_3^*) = (4, 197, 118)$ should be used to maximize the benefit throughput. For the greedy algorithm, it also performs QoS adjustments dynamically when a user departs to further improve the benefit throughput obtainable by the system.

The number of possible cases of $(n_1, n_2, \ldots, n_M)$ from which the optimal set $(n_1^*, n_2^*, \ldots, n_M^*)$ can be found is upper bounded by the number of ways of dividing N_M into M sets subject to Condition 2 above. Thus the time complexity involved in enumerating and applying Eq. (1) and Condition 2 is $O(N_M^{M-1})$. Since it is unlikely for a service provider to store more than 5 versions of the same video title at different QoS levels, e.g., $M < 5$, the time complexity is still manageable in practice.[3] Once we find the best $(n_1^*, n_2^*, \ldots, n_M^*)$ set for each arrival rate λ, we can then build a lookup table recording their relationship, along with the benefit throughput obtainable.

In special cases where the system does not provide differentiated services such that all users demand the same QoS level, say i, our method is still applicable by setting $\forall j, v_j = 0, j \neq i$. Then, the best set found would be $(0, \ldots, 0, n_i^*, 0, \ldots, 0)$, i.e., the system will reserve slots only for QoS level i so as to maximize the benefit throughput. For systems that provide differentiated services, the number of QoS levels, M, is essentially driven by user demands in QoS. In arrival scenarios in which all clients demand at least QoS level i and nothing less, $M = i$ since QoS level $i + 1$ or below is not acceptable to users. The static table generated above can be used by the server to provide "time-differentiated" services, i.e., in "bargain" time periods (say 5 pm–8 am, noon–2 pm), more QoS levels are offered to accommodate more users with a larger M value, while in "business" time periods (say 8 am–noon, 2 pm–5 pm), less QoS levels are offered to guarantee a minimum QoS level with a smaller M value. For example, during a bargain time period, $M = 3$ and during a business time period, $M = 2$. The system can keep a table for $M = 2$ and another one for $M = 3$, giving the optimal resource allocation for $M = 2$ and $M = 3$, respectively, to maximize the benefit throughput obtainable. In this case, the system can switch between $M = 2$ and $M = 3$ by again performing a table lookup operation at run time efficiently for QoS control based on benefit optimization. These bargain and business time periods can be advertised by the service provider so that users are aware of the minimum QoS level guaranteed and the price to pay for the service. Alternatively the service provider can use multiple video servers, each servicing a class of users with a fixed number of QoS levels, e.g., $M = 3$ exclusively all the time.

3.4. Priority-based resource reservation

The baseline and greedy algorithms developed in the paper assume that all users in the system are in the same priority class, i.e., no priority distinction. This results in a single set of benefit and penalty values applied to all users. The concept of QoS control based on benefit optimization can be applied to applications where multiple priority classes of users exist, such that each priority class has a distinct QoS level. For example, if there are three QoS levels ($M = 3$), then three priority classes could exist, each at a distinct QoS level. In this case, a "priority-based QoS control" algorithm can be developed using the same methodology discussed in the paper to find the best partition set $(n_1^*, n_2^*, \ldots, n_M^*)$ such that n_i^* is specifically reserved to serve only users in priority class i with the objective again to optimize the resulting benefit throughput.

We first recognize that unlike the baseline and greedy algorithms designed for dealing with users in a single class, the priority-based QoS control algorithm is designed for servicing users in multiple priority classes. For this reason, we should not compare or contrast the baseline/greedy algorithms with the priority-based algorithm since they don't have the same user basis. For the priority-based QoS control algorithm, users will be arriving from M distinct sources, one for each priority class. Let Λ_i be the arrival rate from class i. Further, although $v_1, v_2, \ldots, v_M$ can be assigned by the service provider to M priority classes (at M QoS levels) in a similar way, a distinct q_i parameter associated with class i will be required to characterize the loss to the system when a user in class i is rejected upon admission. Note that a user in class i is rejected when all n_i^* slots allocated are used up. Also note that n_i^* is reserved based on the knowledge of the arrival rates and benefit/penalty characteristics of users in different priority classes such that the benefit throughput would be optimized, so no "spill-over" into other QoS levels (other priority classes) would be allowed when a rejection occurs.

Below we give a closed-form expression of the benefit throughput $\mathcal{BT}_p$ obtainable under the priority-based QoS control algorithm designed for dealing with users in M multiple classes (corresponding to M QoS levels):

$$\begin{aligned}
&\mathcal{BT}_p(n_1 \dots n_M, \Lambda_1 \dots \Lambda_M, \mu, v_1 \dots v_M, q_1 \dots q_M)\\
&= \sum_{k=1}^{M}\left(\sum_{i=1}^{n_k} i\mu \times v_k \times \frac{\frac{1}{i!}\left(\frac{\Lambda_k}{\mu}\right)^i}{1+\sum_{j=1}^{n_k}\frac{1}{j!}\left(\frac{\Lambda_k}{\mu}\right)^j}\right)\\
&-\sum_{k=1}^{M}\left(\Lambda_k q_k \times \frac{\frac{1}{n_k!}\left(\frac{\Lambda_k}{\mu}\right)^{n_k}}{1+\sum_{j=1}^{n_k}\frac{1}{j!}\left(\frac{\Lambda_k}{\mu}\right)^j}\right)
\end{aligned} \tag{3}$$

The derivation follows the concept that the system behaves as if it contains M separate $m/m/n_i/n_i$ queues, with queue i (class i) now having a distinct, independent input arrival rate Λ_i. Users in class i are served with a departure rate of μ in queue i allocated with n_i slots. When a user departs in queue i, a benefit value of v_i is earned and when a user is rejected by queue i, a penalty of q_i is assessed. Equation (3) can be used to find the the best partition set $(n_1^*, n_2^*, \dots, n_M^*)$ such that $\mathcal{BT}_p$ is maximized. The complexity of the search process and the way the result can be applied are the same as discussed earlier in Section 3.3.

4. Modeling and analysis

In this section, we demonstrate the effectiveness of the baseline and greedy algorithms developed in this paper by means of numerical data and simulation studies. As a basis for comparison, we consider three reference algorithms as follows:

1. Highest-QoS-only (HQO): In this algorithm, we only assign users to the highest QoS level (despite there are M versions for each video), i.e., $(n_1, n_2, \dots, n_M)$ is equal to $(N_1, 0, \dots, 0)$.
2. Lowest-QoS-only (LQO): In this algorithm, we only assign users to the lowest QoS level (despite there are M versions for each video), i.e., $(n_1, n_2, \dots, n_M)$ is equal to $(0, 0, \dots, N_M)$.
3. Equal-Share (ES): In this algorithm, we assign equal amounts of resources to each QoS level, i.e., $(n_1, n_2, \dots, n_M)$ is equal to $(N_1/M, N_2/M, \dots, N_M/M)$.

Comparing with these algorithms, the baseline and greedy algorithms developed will instead perform QoS control based on the best partition of $(n_1, n_2, \dots, n_M)$ as dictated by the input parameter values of λ, μ, $v_1, v_2, \dots, v_M$, and q.

4.1. Disk characteristics and data model

The disk we have selected is a Seagate with 20 G bytes disk space with the average seek time and rotational latency being 16.5 msec, read/write rate 33 MBps, 512 bytes per sector, 64 sectors per track, 4096 tracks per side, and 160 sides on the disk. The disk is filled with video titles randomly and each video has three versions, i.e., the full MPEG 1 format, the H-261 format, and the H-263 format, all of which are from trace files of the movie “Star Wars”. Different movie titles are simulated by using different group of picture (GOP) entry points into the trace files. As the disk is filled, a mapping table is built to map the locations of data stored on disk with playback cycles for various video titles with their different versions in $T_{\mathrm{SR}} = 1$ second increment. We compute the values of N_1 off-line by using classic statistical admission control [5] using the trace data for the full MPEG 1 version at the highest QoS level so that the probability of disk overload does not exceed 10^{-4}. By using the average seek time and rotational latency of 16.5 msec and the read/write rate of 33.3 MBps, it turned out that $N_1 = 55$, that is, 55 video requests can be serviced concurrently so that the probability of disk overload is less than 10^{-4}. Specifically, a three-step procedure is followed to determine $N_1 = 55$ at static time [5]. First, the total number of blocks which the disk is able to access in a playback cycle based on the bandwidth characteristics of the disk is calculated. Second, the probability distribution (a histogram) of the number of disk blocks retrieved by a single video stream in *any* playback cycle is obtained by analyzing the video trace data. Note that the number of disk blocks retrieved in any playback cycle by a video stream is a variable for VBR video, so a probability distribution is used. Third, the probability distribution of N video streams is obtained by performing a convolution of the probability distributions of N video streams. The resulting cumulative probability distribution then is used to calculate the disk overload probability, that is, the probability that the total number of blocks required by N video streams is greater than the total number of blocks the disk is able to access in one playback cycle. For the disk selected in the simulation, it happened that at $N = 56$, the disk overload probability exceeds 10^{-4}, so $N_1 = 55$. We performed a similar procedure for requests being served at the next QoS level, yielding $N_2 = 197$. Lastly, we obtained $N_3 = 552$ for the maximum number of requests that can be served simultaneously at the lowest QoS level.

4.2. *Optimal* (n_1^*, n_2^*, n_3^*) *under the baseline algorithm*

Based on the values of $(N_1 = 55, N_2 = 197, N_3 = 552)$ computed, we consider the arrival rate of streaming requests in the range of [10, 60] arrivals/minute, with 10 arrivals/minute representing a lightly-loaded system and 60 arrivals/minute representing a heavily-loaded system. Thus, based on Eq. (1) and Condition 2 we can easily build a table off-line enumerating the optimal (n_1^*, n_2^*, n_3^*) for each possible arrival rate when the values of v_1, v_2, v_3, and q are given. The values of v_1, v_2, v_3, and q are application dependent. Table 1 below shows the optimal (n_1^*, n_2^*, n_3^*) values together with their associated optimal benefit throughput values, when $v_1 = 20$, $v_2 = 10$, $v_3 = 5$, $q = 1$, $\mu = 0.1$ departures/minute (10 minutes of viewing time per user), and the arrival rate λ varies in the range of [10, 60] in increment of 5 arrivals/minute. As expected, the benefit throughput obtained at the optimal (n_1^*, n_2^*, n_3^*) setting increases as λ increases because the system goes from lightly loaded to heavily loaded (i.e., more rewards are added due to a higher system throughput); however, as λ exceeds a threshold, the benefit throughput deteriorates because many clients are rejected (i.e., more penalties incur due to a higher rejection rate).

4.3. Numerical data

Figures 1–5 compare the benefit throughput (the Y-coordinate) obtainable by the baseline, highest-QoS-only (HQO), lowest-QoS-only (LQO), and equal-share (ES) algorithms as a function of the user arrival rate (the X-coordinate) when given a set of input parameter values on v_1, v_2, v_3 and q. First we note that the unit of the y-coordinate is "value-units/sec," so even a difference of 0.1 can be significant, e.g., it can correspond to a difference of 0.1 dollar/sec to a service provider. Figure 1 shows the extreme case in which $v_1 \gg v_2, v_3$. In this case, the baseline algorithm behaves like the HQO algorithm because when the benefit value at the highest QoS level is much higher than others, reserving all resources at the highest

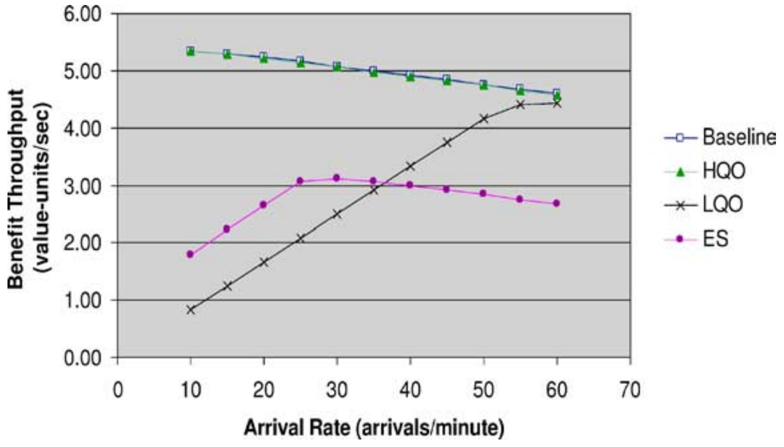


Figure 1. Comparing the benefit throughput obtainable: ($v_1 = 60, v_2 = 10, v_3 = 5, q = 1$).

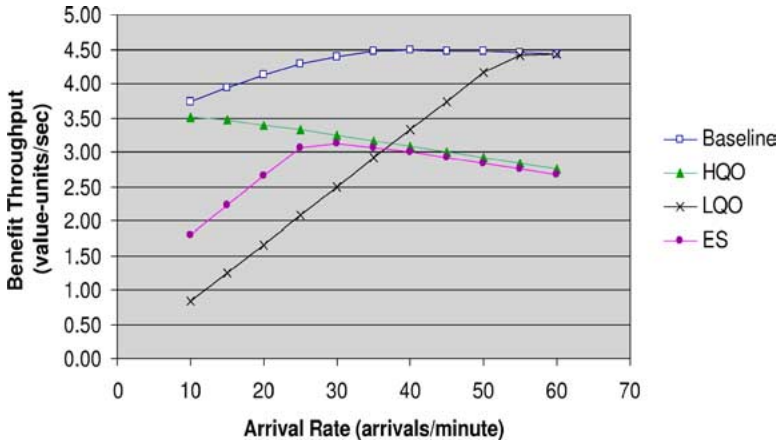


Figure 2. Comparing the benefit throughput obtainable: ($v_1 = 40, v_2 = 10, v_3 = 5, q = 1$).

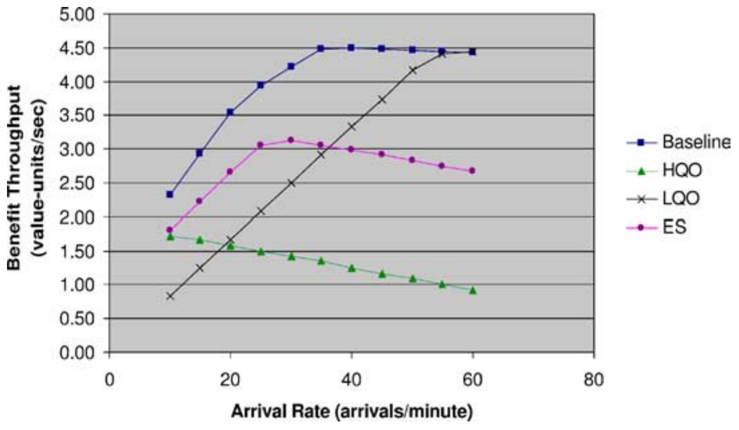


Figure 3. Comparing the benefit throughput obtainable: ($v_1 = 20, v_2 = 10, v_3 = 5, q = 1$).

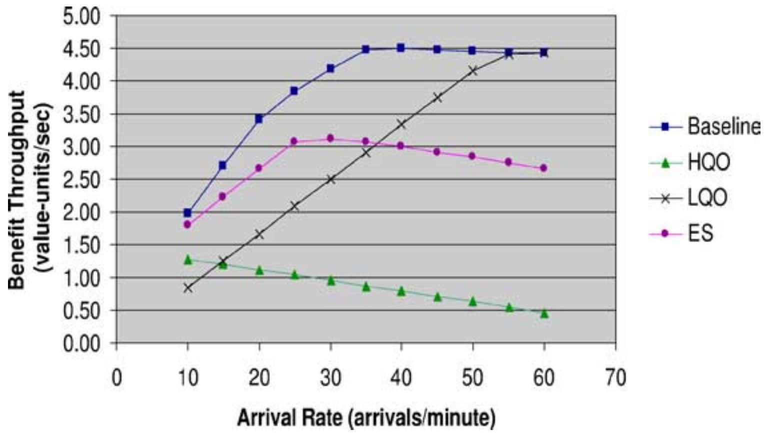


Figure 4. Comparing the benefit throughput obtainable: ($v_1 = 15, v_2 = 10, v_3 = 5, q = 1$).

QoS level would yield the best benefit throughput. On the other hand, figure 5 shows the extreme case in which $v_1 = v_2 = v_3$. In this case, the baseline algorithm behaves like the LQO algorithm, because when the benefit value is the same at all QoS levels, reserving all resources at the lowest QoS level will allow more users to be admitted and thus a higher overall benefit received by the system.

Figures 2–4 show three more general cases in which $v_1 > v_2 > v_3$ holds true but not necessarily $v_1 \gg v_2, v_3$. These three cases exhibit a similar trend that the benefit throughput obtainable by the baseline QoS control algorithm is much better than the three reference QoS control algorithms. We see that always serving users at the highest QoS level (under the HQO algorithm) will result in a heavy loss of benefit to the system except when the benefit value of the highest QoS level is much higher than those of other QoS levels (as in figures 1

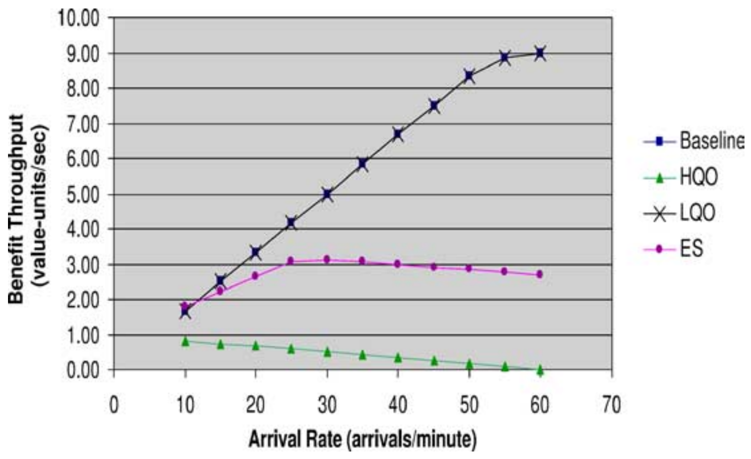


Figure 5. Comparing the benefit throughput obtainable: ($v_1 = 10, v_2 = 10, v_3 = 10, q = 1$).

and 2), or when the system is very lightly loaded. This means that an algorithm based on not reserving resources at all for lower QoS levels is unlikely to perform well since when the traffic is modest, the system may unnecessarily reject users since resources can be quickly used up by users at the highest QoS level. On the other hand, an algorithm always serving users at the lowest QoS level (under the LQO algorithm) can perform well only when the benefit value of the lowest QoS level is about the same as those at other levels (as in figure 5), or when the system is heavily loaded, in which case the system avoids rejecting too many users by serving all users at the lowest QoS level. When the system is lightly loaded, in general the LQO algorithm would yield the worst benefit throughput among all since the net benefit of successfully completing user services at the lowest QoS level is marginal. The ES algorithm falls within these two extremes with the benefit throughput obtainable being modest all the time. However, since the resource allocation to M QoS levels is equal and indiscriminating in nature, it performs poorly in all cases compared with the baseline algorithm which allocates resources discriminatively with the goal to maximize the benefit throughput.

4.4. Simulation validation

We developed a single-disk simulator based on the disk characteristics and data model described earlier to validate the design of the baseline and greedy algorithms and to see the performance gain of the greedy algorithm compared with the baseline algorithm. In the simulation, we periodically (every 20 minutes) change the arrival rate of requests by an increment of 5 arrivals/minute, starting from an initial arrival rate of 10 arrivals/minute. This change is detected as an event to which the system responds by changing to another (n_1^*, n_2^*, n_3^*) set based on Table 1 for admission control of the three QoS levels. While the system is under a (n_1^*, n_2^*, n_3^*) set, it admits and rejects users, as well as scheduling requests, in accordance with the baseline and greedy QoS control algorithms described in Section 3.

Again the departure rate of requests is assumed to be 0.1 departure/minute. All stream-requests admitted are served based on the SCAN algorithm in each $T_{SR} = 1$ second cycle, that is, all admitted requests are arranged in a scheduled queue in every cycle based on the disk locations of data blocks to be accessed so that the disk read/write heads only need to scan the disk in one direction once to collect all the data needed for all streams so as to minimize the seek time overhead. Any stream not having the needed data retrieved prior to its playback instant suffers a data loss, the probability of which is also a performance measure to be collected in the simulation.

4.4.1. Simulation results and physical interpretations. Recall that we assume that there is plenty of buffer space, so the disk bandwidth is the only bottleneck that can cause data loss. From all the simulation runs we have covered, we observe little data loss under all algorithms considered. This is expected since all algorithms considered are based on resource reservation and (statistical) admission control.

Figure 6 compares the benefit throughput $\mathcal{BT}$ obtained under the greedy algorithm vs. that obtained under the baseline algorithm, as a function of the arrival rate. Each interval covers 20 minutes of simulation time, at the end of which the benefit throughput obtained in that interval is computed.

We observe that the greedy algorithm always performs better than the baseline algorithm in terms of the total benefit throughput it brings to the system, due to the employment of dynamic QoS adjustments. Note that in the first few intervals of figure 6 during which the system is lightly-loaded, the system mostly serves clients at the highest or the medium QoS level so as to increase the benefit received due to departing clients, while at the last few intervals during which the system is heavily loaded, the system serves clients mostly at the medium or the lowest QoS level so as not to reject too many clients and not to diminish the benefit received. The greedy algorithm, in addition to following this QoS control strategy as the baseline algorithm, also effectively adds more benefit values to the system by raising QoS levels of existing users whenever a higher QoS level client departs. As a result, it always performs better than the baseline algorithm.

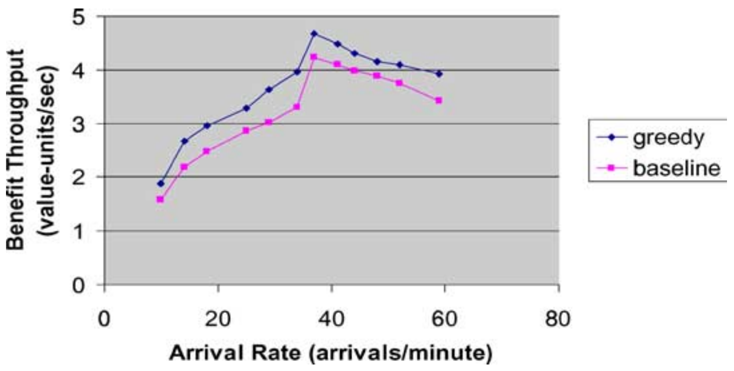


Figure 6. Comparing baseline and greedy algorithms in $\mathcal{BT}$.

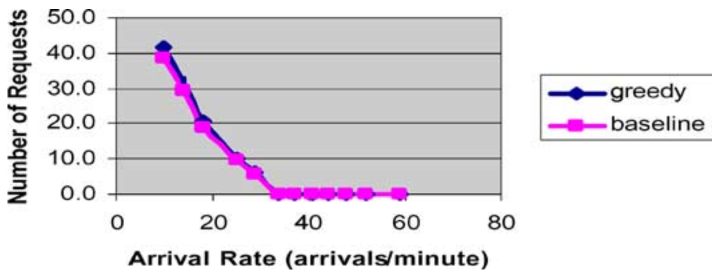


Figure 7. Population of high-QoS clients.

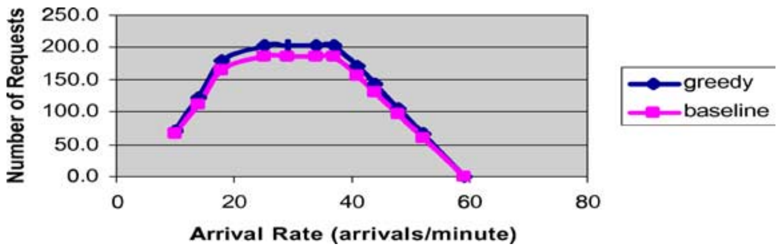


Figure 8. Population of medium-QoS clients.

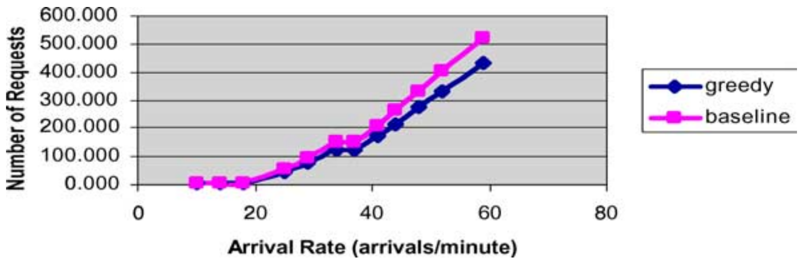


Figure 9. Population of low-QoS clients.

Figures 7–9 compare the average numbers of requests at the highest, medium and lowest QoS levels, respectively, under the baseline and greedy algorithms, also as a function of the arrival rate in increment of about 5 arrivals/minute. Figures 7 and 8 indicate that the greedy algorithm is able to serve more clients at the highest or medium QoS levels compared with the baseline algorithm especially when the load is modest. This is due to the fact that the greedy algorithm moves lower-QoS clients to a higher-QoS level greedily whenever a higher-QoS client departs. Consequently, there are fewer clients being served at the lowest QoS level under the greedy algorithm when compared with the baseline algorithm, as demonstrated in figure 9. Allowing more clients to be served at higher QoS levels in our greedy algorithm has the direct effect of improving the overall benefit throughput obtainable by the system since when a user departs, the benefit it brings to the system is proportional to the QoS level received. It should be mentioned that both the baseline and greedy algorithms would yield the same rejection rate and system throughput (number of users completed per time unit). Like the baseline algorithm, the greedy algorithm also uses the the optimal (n_1^*, n_2^*, n_3^*) set for QoS control. Thus, moving users from lower QoS levels to higher QoS levels upon user departures by the greedy algorithm will not change the total number of users admissible by the system.

5. Conclusion

We have proposed and analyzed a class of QoS control algorithms applicable to video servers that support the notion of QoS negotiation and renegotiation. Our design goal is to optimize the *benefit throughput* as a result of servicing clients at different QoS levels. We designed baseline and greedy QoS control algorithms based on the concept of resource reservation so that resources are reserved a priori for different QoS levels based on the workload to the system. The system can admit clients based on the allocation. These algorithms can adapt to workload changes at run time by dynamically re-allocating resources reserved to requests at different QoS levels by performing a simple table lookup operation (e.g., Table 1), so that the benefit throughput can be maximized. Further, for systems that allow the QoS level to be raised (but not lowered) during the service lifetime of users, the greedy algorithm can improve the benefit throughput further by performing QoS adjustments, as opposed to the baseline algorithm designed for systems that do not allow QoS changes at run time. We validated the greedy algorithm design with simulation and demonstrated that the greedy algorithm outperforms other algorithms in terms of the benefit throughput without sacrificing the performance of the system, such as the throughput and rejection probability.

A future research area is to investigate if the algorithm designed in this paper can be integrated seamlessly with existing QoS-based cost models in implementing web-based video servers that support differentiated QoS services and pricing.

Notes

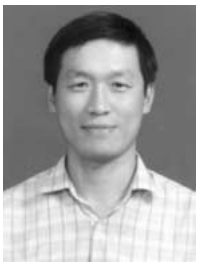
1. Later in Section 3.4, we will discuss how to apply the concept of benefit optimization to priority-based QoS control.
2. The same QoS control is applied to all users in the system since there is no priority distinction among users, i.e., only a single priority class exists. Later in Section 3.4 we will treat the case in which multiple priority classes of users exist, with each priority class corresponding to a distinct QoS level.
3. Note that our method proposed in the paper does not prevent $M \geq 5$ although in general we would not expect $M \geq 5$ because of excessive storage requirements.

References

1. C. Aurrecoechea, A.T. Campbell, and L. Hauw, “A survey of QoS architectures,” ACM/Springer Multimedia Systems, Vol. 6, No. 3, pp. 138–151, 1998.

2. S. Brandt, G. Nutt, T. Berk, and M. Humphrey, "Soft real-time application execution with dynamic quality of service assurance," in 6th International Workshop on Quality of Service, Napa, CA, May 1998.
3. S.T. Cheng, C.M. Chen, and I.R. Chen, "A study of self-adjusting quality of service control schemes," in 1998 Winter Simulation Conference, Washington D.C., 1998, pp. 1623–1628.
4. S.T. Cheng, C.M. Chen, and I.R. Chen, "Dynamic quota-based admission control with subrating in multimedia servers," ACM/Springer Journal on Multimedia Systems, Vol. 8, No. 2, pp. 83–91, 2000.
5. E. Chang and A. Zakhor, "Cost analyses for VBR servers," IEEE Multimedia, Vol. 3, No. 4, pp. 56–71, Winter 1996.
6. T. Henderson, J. Crowcroft, and S. Bhatti, "Congestion pricing," IEEE Internet Computing, Vol. 5, No. 2, 2001, pp. 85–89.
7. L. Kleinrock, Queueing Systems, Vol. 1: Theory, John Wiley and Sons, 1975.
8. G. Lawton, "Video streams into the mainstream," IEEE Computer, July 2000, pp. 12–17.
9. W. Lee and B. Sabata, "Admission control and QoS negotiation for soft-real time applications," in IEEE International Conference on Multimedia Computing and Systems, Vol. 1, Florence, Italy, 1999, pp. 147–152.
10. B. Li and K. Nahrstedt, "A control-based middleware framework for QoS adaptation," IEEE Journal of Selected Area in Communications, Vol. 17, No. 9, pp. 1632–1650, 1999.
11. G.J. Nutt et al., "Dynamically negotiated resource management for data intensive application suites," IEEE Trans. Knowledge and Data Engineering, Vol. 12, No. 1, pp. 78–95, 2000.
12. P.J. Shenoy and H.M. Vin, "Cello: A disk scheduling framework for next generation operating systems," in 7th ACM Inter. Conf. on Measurement and Modeling of of Computer Systems (SIGMETRICS '98), Madison, 1998, pp. 44–55.
13. T.-P.J. To and B. Hamidzadeh, "Run-time optimization if heterogeneous media access in a multimedia server," IEEE Transactions on Multimedia, Vol. 2, No. 1, pp. 49–61, 2000.
14. R. Wijayaratne and A.L.N. Reddy, "Integrated QoS management for disk I/O," in IEEE International Conference on Multimedia Computing and Systems, Vol. 1, Florence, Italy, 1999, pp. 487–492.

Ing-Ray Chen received the BS degree from the National Taiwan University, Taipei, Taiwan, and the M.S. and Ph.D. degrees in computer science from the University of Houston. He is currently an associate professor in the Department of Computer Science at Virginia Tech. His research interests include mobile computing, pervasive computing, multimedia, distributed systems, real-time intelligent systems, and reliability and performance analysis. Dr. Chen has served on the program committee of numerous conferences, including as program chair for 29th IEEE Annual International Computer Software and Application Conference in 2005, 14th IEEE International Conference on Tools with Artificial Intelligence in 2002, and 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology in 2000. Dr. Chen currently serves as an Associate Editor for *IEEE Transactions on Knowledge and Data Engineering*, *The Computer Journal*, and *International Journal on Artificial Intelligence Tools*. He is a member of the IEEE/CS and ACM.



Sheng-Tun Li is an Associate Professor in the Institute of Information Management, National Cheng Kung University, Tainan, Taiwan. He received his Ph.D. in Computer Science from the University of Houston in 1995. His research interests include multimedia streaming, artificial intelligence, and Java computing. Dr. Li is a member of the IEEE/CS and ACM.

I-Ling Yen received her BS degree from Tsing-Hua University, Taiwan, and her MS and PhD degrees in Computer Science from the University of Houston. She is currently an Associate Professor of Computer Science at the University of Texas at Dallas. Dr. Yen's research interests are in distributed systems, fault-tolerant computing, self-stabilization algorithms, and security. She has served as program co-chair for the 1997 IEEE High Assurance Systems Engineering Workshop, the 1999 IEEE Symposium on Application-Specific Systems and Software Engineering Technology, and the 1999 Annual IEEE International Conference on Computer Software and Applications Conference. Dr. Yen is a member of the IEEE/CS.