Optimal Quantization of Periodic Task Requests on Multiple Identical Processors

Laura E. Jackson, *Student Member*, *IEEE*, and George N. Rouskas, *Senior Member*, *IEEE*

Abstract—We simplify the periodic tasks scheduling problem by making a trade off between processor load and computational complexity. A set N of periodic tasks, each characterized by its density ρ_i, contains n possibly unique values of ρ_i. We transform N through a process called *quantization*, in which each $\rho_i \in N$ is mapped onto a service level $s_j \in L$, where $|L| = l \ll n$ and $\rho_i \leq s_j$ (this second condition differentiates this problem from the *p*-median problem on the real line). We define the Periodic Task Quantization problem with Deterministic input (PTQ-D) and present an optimal polynomial time dynamic programming solution. We also introduce the problem PTQ-S (with Stochastic input) and present an optimal solution. We examine, in a simulation study, the trade off penalty of excess processor load needed to service the set of quantized tasks over the original set, and find that, through quantization onto as few as 15 or 20 service levels, no more than 5 percent processor load is required above the amount requested. Finally, we demonstrate that the scheduling of a set of periodic tasks is greatly simplified through quantization and we present a fast online algorithm that schedules quantized periodic tasks.

Index Terms—Multiprocessor scheduling, periodic tasks scheduling, quantization.

1 Introduction

1.1 The Periodic Tasks Scheduling Problem

We are given a number $m \geq 1$ of processors and a number $n > m$ of periodic real-time tasks. A periodic task is made up of an infinite number of subtasks, each of length one. Time is slotted such that a processor can process one subtask in one slot. Associated with each task is a rational density ρ_i, $0 < \rho_i < 1$, which represents the task's demand for processing time, in terms of subtasks per slot. If ρ_i is written as a fraction in lowest terms, $\rho_i = \frac{C_i}{D_i}$, then the numerator C_i is the computation time (number of subtasks) that must be processed within the period D_i. A *job* of a task refers to the collection of C_i subtasks that must be completed within one period D_i. All subtasks of the first job of task i are released for processing at time 0 and must be completed by time D_i; subtasks of the kth job are released for processing at time $(k-1)D_i$ and must be completed by time kD_i. A subtask may receive processing time from any of the m identical processors. Scheduling is preemptive so that subsequent subtasks need not be processed consecutively, nor even by the same processor.

The system is subject to two constraints: The *Processor Constraint* requires that, at any instant in time, a processor may work on at most one subtask, and the *Task Constraint* requires that, at any instant in time, a task may have a subtask being processed by at most one processor. This model is nearly identical to that considered in [3] and [5], which does not require the ratio $\frac{C_i}{D_i}$ to be in lowest terms. Any problem instance from their model becomes a problem instance for our model by setting $\rho_i = \frac{C_i}{D_i}$; any feasible schedule for our problem is also feasible for theirs. However, there exist feasible schedules for their problem that are not feasible for ours. For example, consider a task with $C_i = 2$ and $D_i = 4$. A feasible schedule according to [3] and [5] may process the two subtasks at any time on the interval $[0, 4)$. Our model, however, views this task as having $\rho_i = \frac{2}{4} = \frac{1}{2}$, and a feasible schedule must process the first subtask on the interval $[0, 2)$ and the second on $[2, 4)$.

1.2 Periodic versus P-Fair Schedules

A more stringent requirement than periodicity is *proportional fairness* or p-fairness [3], [4], [2]. Intuitively, a p-fair schedule closely mimics the idealized fluid system, in which both time and the jobs of a task are infinitely divisible, in contrast to the integer time slot and unit-length subtask restrictions. A p-fair schedule meets the requirements of periodicity, but a periodic schedule will not necessarily be p-fair. As it turns out, the periodic tasks scheduling problem is most quickly solved by trying to create a p-fair schedule.

The problem of finding a p-fair (and, hence, periodic) schedule for the periodic tasks scheduling problem has been solved; in [3], Baruah et al. present Algorithm PF which, at each time slot, runs in time that is linear in the size of the input in bits. In [4], Baruah et al. give a faster algorithm PD with time complexity $O(\min\{m \lg n, n\})$ at each slot. Finally, in [2], Anderson and Srinivasan simplify the priority definition used by PD to yield PD^2, with time complexity $O(m \lg n)$ at each slot. The priority of competing subtasks is determined in part by a subtask's slot deadline, the latest slot in which it may be scheduled while still maintaining the p-fairness of the schedule.

While algorithms PF, PD, and PD^2 are all optimal in that they always find a p-fair (and, hence, periodic) schedule whenever one exists (i.e., whenever $\sum_{i=1}^{n} \rho_i \leq m$), the computation time per slot is a function of the number of tasks n. In particular, since the algorithms at each slot select

• *The authors are with the Department of Computer Science, North Carolina State University, Box 7534, Raleigh, NC 27695-7534. E-mail: lojack@anr.mcnc.org or lejacks2@eos.ncsu.edu; rouskas@csc.ncsu.edu.*

Manuscript received 9 Jan. 2002; revised 5 Nov. 2002; accepted 14 Feb. 2003. For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 115673.

the m tasks with the most imminent slot deadlines for processing, the running time per slot can be no lower than $O(m \lg n)$. Therefore, these algorithms may not be appropriate for applications with a very large number of tasks. For instance, consider a Web server for a popular Web site which uses multiple processors to serve client requests. Such a Web site may receive millions of requests per minute, and therefore it is essential to have a scheduling algorithm with a running time independent of the number of requests. Also, consider the recent announcement by IBM regarding the creation of server farms that will provide processing power to applications on demand. This view of processing power as a service that is provided by some form of public utility may be appealing to both individuals and companies of all sizes (which may wish to reduce costs by outsourcing their computation needs much like they "outsource" their power or water needs). Such a public utility will face very large task sets that are also highly dynamic in nature. Thus, it will have to rely on fast scheduling algorithms in order to provide service in an effective and efficient manner.

We propose to simplify the scheduling algorithms in multiprocessor systems by restricting the number of service levels offered. Operating a multiprocessor system that provides only a (small) set of quantized service levels makes sense in many respects. In such a system, many functions, such as billing and the scheduling, management, and handling of dynamic task requests, will be significantly simplified as compared to a system offering a continuous spectrum of rates. On the other hand, limiting the number of supported rates does have a disadvantage in that it may require more processing power than a continuous-rate system to accommodate a given set of task requests. Specifically, rather than receiving the exact rate needed, a task may have to subscribe to the next higher rate offered by the system. As a result, quantization will have an adverse effect on performance, which will manifest itself either as a higher blocking probability (i.e., a higher probability of denying a task request compared to a continuous-rate system), or as a lower utilization (since a larger number of processors may be needed to carry the same set of tasks).

Our goal is to determine the set of service levels that strikes a balance between the two conflicting goals of simplicity and performance. Specifically, we address the issue of determining the optimal set of rates in which to quantize the processor power given 1) a fixed set of task requests, or 2) the probability density function of task requests. The objective is to minimize the performance penalty due to quantization, i.e., the difference between the amount of processor power required by a system supporting an optimal set of quantized rates and that required by a continuous-rate system.

Similar work in the context of ATM networks [9] has demonstrated that ATM networks offering a handful of quantized levels suffer little performance degradation compared to continuous rate networks. Our conclusions are similar, although our approach is different and our results are stronger. Specifically, [9] takes a queueing theoretic approach, considers a single link with Poisson arrivals, and uses a heuristic technique (simulated annealing) to obtain a suboptimal vector of service levels. In this paper, we use a dynamic programming approach which allows us to compute the *optimal* service levels in a very efficient manner.

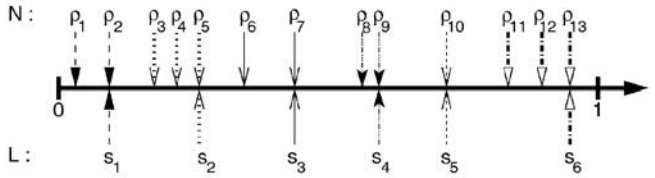

Fig. 1. Sample mapping of task densities to service levels.

Our problem bears some resemblance to the problems considered in [7] and the references therein (e.g., the p-median problem and the simple uncapacitated plant location problem on the real line). These problems allow a task requesting a rate ρ to be served by the nearest service level, whether it is above or below ρ. Our problem is fundamentally different in that we require a task to receive *at least* the rate requested. To our knowledge, this problem has never previously been treated in the literature.

The rest of the paper is structured as follows: We define the Periodic Task Quantization problem with Deterministic input (PTQ-D) in Section 2 and, then, present an optimal dynamic programming solution. The trade off penalty in terms of excess processor load is examined in a simulation study given in Section 2.3. In Section 3, we introduce the Periodic Task Quantization problem with Stochastic input (PTQ-S), for which the input is a probability density function of task requests. We present an optimal solution for a certain class of probability density functions, as well as an approximate solution. Last, we consider the scheduling of a quantized set of periodic tasks, and give a new algorithm which runs in time $O(m)$ at each time slot.

2 The Periodic Task Quantization Problem with Deterministic Input (PTQ-D)

2.1 Statement of the Problem

Let N be a set of n periodic task densities $\{\rho_1, \ldots, \rho_n\}$, such that $\rho_1 \leq \rho_2 \leq \cdots \leq \rho_n$ and let the density of N be $\rho_N = \sum_{i=1}^{n} \rho_i$. A set $L = \{s_1 \ldots s_l\}$, $s_1 < s_2 < \cdots < s_l$, $1 \leq l \leq n$, is a *feasible quantization set* of N if and only if $\rho_i \leq s_l$, $i = 1 \ldots n$. For notational convenience, we assume $s_0 = 0$. Associated with a feasible quantization set is an *implied mapping* from $N \rightarrow L$, where $\rho_i \rightarrow s_j$ if and only if $s_{j-1} < \rho_i \leq s_j$. That is, a periodic task that requests a share of processor power equal to ρ_i will be able to meet its periodic deadlines if it is given a share of processor power (or service level) equal to s_j, so long as $\rho_i \leq s_j$. Fig. 1 shows a sample mapping from a task set of 13 densities onto a quantization set of six service levels. Let N_j be the set of tasks mapped to service level s_j and let $|N_j| = n_j$.

Problem 1. *(PTQ-D) Given a set N of n periodic task densities, $\rho_1 \leq \rho_2 \leq \cdots \leq \rho_n$, find a feasible set L of l quantized service levels, $s_1 < s_2 < \cdots < s_l$, $1 \leq l \leq n$, which minimizes the following objective function:*

$$g_D(s_1,\ldots,s_l) = \sum_{j=1}^{l}\sum_{\rho_i\in N_j}(s_j-\rho_i) \quad (1)$$

$$= \sum_{j=1}^{l}(n_j s_j) - \rho_N \quad (2)$$

$$= q_D(s_1,\ldots,s_l) - \rho_N. \quad (3)$$

The objective function g_D represents the penalty of excess processor load used by the quantized set above that requested by the original task set. Processor capacity is a limited resource and, therefore, minimizing wasted processor load will allow the system to accept more customers and serve those customers in a more cost-efficient way. The second term of (3), ρ_N, the *requested load*, is the amount of processor load requested by the original task set, while the first term, $q_D(s_1,\ldots,s_l)$, is the *quantization load*, the processor load assigned to the set of quantized tasks.[1] The minimum (optimal) value of g_D is called g_D^*, and a feasible set L at which g_D^* is obtained is called an *optimal* quantization set of N. Minimizing g_D also minimizes a quantity called the Normalized Quantization Load for deterministic input, NQL_D:

$$NQL_D = \frac{q_D(s_1\ldots s_l)}{\rho_N} \quad (4)$$

$$= \frac{\sum_{j=1}^{l} n_j s_j}{\rho_N} \quad (5)$$

$$\geq 1. \quad (6)$$

Clearly, the closer NQL_D is to 1, the fewer processor resources are wasted.

For any feasible quantization set for which $\rho_n < s_l$, the objective function g_D can be reduced by setting $s_l = \rho_n$. Therefore, in an optimal quantization set, the maximum service level s_l must equal the maximum task density ρ_l. Furthermore, we can state the following lemma:

Lemma 1. *Let N be a set of n periodic task densities such that $\rho_1 \leq \rho_2 \leq \cdots \leq \rho_n$. There exists an optimal quantization set $L = \{t_1,\ldots,t_l\}$, $t_1 < t_2 < \cdots < t_l$, of N, for which $t_j \in N$, for each $j = 1,\ldots,l$.*

Proof. Suppose there exists an optimal quantization set of N called $L_1 = \{s_1\ldots s_l\}$, $s_1 < s_2 < \cdots < s_l$, for which there exists some $s_a \in L_1$, but $s_a \notin N$. There can be no ρ_i that is mapped to s_a. If there were, then s_a could be moved down to $s_a - \Delta$, for some $\Delta > 0$, and the objective function could be lowered, contradicting the optimality of L_1. Therefore, we can create L from L_1 by setting $t_j = s_j$ for $j \neq a$ and $t_a = \rho_n$. □

2.2 Dynamic Programming Solution: Algorithm Quantize

We now present the algorithm Quantize which uses a dynamic programming approach to obtain an optimal set of service levels for problem PTQ-D. This approach is based on the observation that, due to Lemma 1, an optimal set of service levels is a subset of the set N of task densities. Quantize computes this optimal subset in an efficient manner.

Quantize builds four tables of values described below and, in doing so, finds the optimal value of the objective function g_D^* and an optimal quantization set L. The $n \times n$ tables `Diff` and `Cumul` hold differences and cumulative sums of differences, respectively; these values will be used to fill in the entries of the $n \times l$ table `Opt`. Entries in `Diff` and `Cumul` are calculated according to the following formulas (the array `rho` holds the elements of the task set N):

$$\texttt{Diff}[i][j] = \texttt{rho}[j] - \texttt{rho}[i]\ ,\ i \leq j$$

$$\texttt{Cumul}[i][j] = \sum_{k=j}^{i}\texttt{Diff}[k][i]$$

$$= \sum_{k=j}^{i}(\texttt{rho}[i] - \texttt{rho}[k])\ ,\ j \leq i.$$

Filling in a single entry of `Opt` corresponds to solving one instance of PTQ-D; entry (i, j) holds the minimum value of the objective function g_D for an instance of PTQ-D in which $N = \{\rho_1,\ldots,\rho_i\}$ and $l = j$. Each entry of `Opt` is calculated recursively, using entries representing smaller problem instances; that is, an instance having a smaller value of n or a smaller value of l or both. Specifically:

$$\texttt{Opt}[i][j] = \begin{cases} 0 & \text{if } i = j \\ \texttt{Cumul}[i][j] & \text{if j} = 1 \\ \min_{k=j-1}^{i-1}\{\texttt{Opt}[k][j-1] + \texttt{Cumul}[i][k+1]\} & j < i \text{ and } j \neq 1. \end{cases} \quad (7)$$

Last, the $n \times l$ table `Prev` holds the information needed to construct the optimal quantization set L. By Lemma 1, each service level $s_j \in L$ must take on the value of some $\rho_i \in N$. `Prev` $[i][j]$ holds the index into the `rho` array of s_{j-1}, assuming that $s_j = \rho_i$; that is, if $s_j = \rho_i$, then $s_{j-1} =$ `rho`[`Prev`$[i][j]$]. `Prev` $[i][j]$ also equals the value of k at which the minimum was attained in the third line of (7) (or $i - 1$ if $i = j$). Note that, for $j = 1$, `Prev` $[i][j]$ is undefined since there is no s_0. Thus, when Quantize finishes building `Prev`, the optimal quantization set L can be constructed using only a few lines of code. The pseudocode description of Quantize can be found in [8].

2.2.1 Correctness Proof

Theorem 1 below proves the correctness of Quantize by demonstrating that the value calculated for `Opt`$[n][l]$ is equal to the optimal value of the objective function g_D^* for an instance of PTQ-D in which a set of n tasks are optimally quantized into l service levels. We first prove two lemmas to aid in the proof of Theorem 1.

Lemma 2. `Opt`$[n][l] = g_D^*(s_1,\ldots,s_l)$ *whenever* $n = l$.

Proof. Since there are as many service levels as there are tasks, each task is assigned exactly the service level it has requested, namely, $s_i = \rho_i$ for $i = 1,\ldots,n$. Thus,

1. The subscript "D" in $g_D(s_1,\ldots,s_l)$ and $q_D(s_1,\ldots,s_l)$ stands for deterministic input. We will derive similar expressions for stochastic input in Section 3.

$g_D^*(s_1, \ldots, s_l) = \sum_{i=1}^{n}(s_i - \rho_i) = 0$, which agrees with the first case of (7). □

Lemma 3. $\texttt{Opt}[n][l] = g_D^*(s_1, \ldots, s_l)$ *whenever* $l = 1$.

Proof. Since there is only one service level, then $s_1 = \rho_n$ and, thus, $g_D^*(s_1) = \sum_{i=1}^{n}(\rho_n - \rho_i) = \texttt{Cumul}[n][1]$, which agrees with the second case of (7). □

Theorem 1. $\texttt{Opt}[n][l] = g_D^*(s_1, \ldots, s_l)$, *for* $n \geq 1$ *and* $1 \leq l \leq n$.

Proof. By induction. For the base case, let $n = 1$ and $l = 1$. By Lemma 2, $\texttt{Opt}[1][1] = g_D^*(s_1)$.

Assume $\texttt{Opt}[i][j] = g_D^*(s_1, \ldots, s_n)$, for all possible (i, j)-pairs $i = 1, \ldots, n$ and $j = 1, \ldots, l$, for some $n \geq 1$ and $1 \leq l < n$. We now prove that $\texttt{Opt}[n][l+1] = g_D^*(s_1, \ldots, s_{l+1})$. Note that this is sufficient for the induction step; by Lemma 3, we can always fill in the first element of each row of the Opt table. Then, we need only fill in each row from left to right, beginning with row 1 and proceeding to row 2, etc. Thus, for the induction step, it is sufficient to show that we can accurately calculate the next entry (one column to the right) in the current row, namely $\texttt{Opt}[n][l+1]$.

Case 1: $l + 1 = n$. By Lemma 2,

$$\texttt{Opt}[n][l+1] = g_D^*(s_1, \ldots, s_l).$$

Case 2: $l + 1 < n$. The largest service level must equal the largest task density: $s_{l+1} = \rho_n$. We next examine all possible values for s_l and choose the one that yields the lowest value of the objective function $g_D(s_1, \ldots, s_{l+1})$. According to Lemma 1, we need only consider the task densities as possible values for s_l; in particular, s_l may only equal one of the following densities: $\{\rho_l, \rho_{l+1}, \ldots, \rho_{n-1}\}$. Suppose $s_l = \rho_k$, for some $k \in \{l, l+1, \ldots, n-1\}$. Then, the tasks $\{\rho_{k+1}, \ldots, \rho_n\}$ will be mapped to $s_{l+1} = \rho_n$, and the contribution to the objective function $g_D(s_1, \ldots, s_{l+1})$ from the s_{l+1}-mapping alone will be:

$$\begin{aligned}\sum_{i=k+1}^{n}(s_{l+1} - \rho_i) &= \sum_{i=k+1}^{n}(\rho_n - \rho_i) \\ &= \sum_{i=k+1}^{n}\texttt{Diff}[i][n] \\ &= \texttt{Cumul}[n][k+1].\end{aligned}$$

This quantity, $\texttt{Cumul}[n][k+1]$, is exactly the second term inside the min function of the third case of (7). Next, we calculate the contribution to the objective function $g_D(s_1, \ldots, s_{l+1})$ from the task mappings to the remaining service levels s_1 to s_l; this contribution depends on the placement of s_1 to s_{l-1} (recall that we have fixed s_l at ρ_k). By the inductive hypothesis, we have already calculated the optimal placement of s_1 to s_{l-1} whenever $s_l = \rho_k$; in particular, $\texttt{Opt}[k][l]$ is the minimum value of the objective function for a problem instance in which $N = \{\rho_1, \ldots, \rho_k\}$ is quantized into l service levels, and the Prev table holds the positions of the service levels that yield this minimum value. Thus, the min function of the third case of (7) does the following: For each possible position ρ_k for s_l, $\rho_k \in \{\rho_l, \ldots, \rho_{n-1}\}$, it calculates the objective function $g_D(s_1, \ldots, s_{l+1}) = \texttt{Opt}[k][l] + \texttt{Cumul}[n][k+1]$ and, then, chooses the position that yields the minimum. The chosen value of k is stored in the Prev table. □

2.2.2 Analysis of Quantize

The operation of Quantize can be divided into four sequential tasks. First, the algorithm builds the Diff table in time $O(n^2)$. Second, it uses the Diff table to fill in the entries of the Cumul table, also in time $O(n^2)$. Third, the algorithm builds the Opt table: For an entry calculated using the third line of (7), the min operation inspects at most n sums, hence, the line with the min operation requires time $O(n)$. There are at most nl entries in Opt and, thus, Quantize builds the Opt table in time $O(n^2 l)$. The fourth and final task of Quantize consists of constructing the optimal quantization set L from the information held in the Prev table, which is accomplished in time $O(l)$. Therefore, the overall running time of Quantize is $O(n^2 + n^2 + n^2 l + l)$ or $O(n^2 l)$.

2.3 Quantifying the Performance Penalty Due to Quantization

2.3.1 Simulation Set-Up and Input Parameters

To determine the penalty in terms of excess processor load resulting from quantization, a simulation study was designed using a variety of different types of task sets N. In particular, six different input distributions were used to generate task sets N in the simulations: uniform, triangle, increasing, decreasing, unimodal, and bimodal. Fig. 2 shows the graph of each input distribution's probability density function; the mathematical expressions of each pdf and cdf are given in [8]. From each input distribution, 100 task sets with $n = 100$ were generated and another one hundred task sets with $n = 1,000$ were generated. Each task set was generated starting from a unique seed for a Lehmer random number generator with modulus $2^{31} - 1$ and multiplier 48,271.

Each task set is then served as input to the algorithm Quantize. We used the normalized quantization load NQL_D (defined in (4)) as the measure of the performance penalty due to quantization. For each task set, NQL_D was calculated for a variety of values of l, the number of service levels in the optimal quantization set.

2.3.2 Simulation Results

Fig. 3 contains six graphs, one for each input distribution. Each graph shows NQL_D along the y-axis corresponding to values of l ranging from $l = 2, 3, \ldots, 100$ along the x-axis, for task sets of size $n = 100$ and $n = 1,000$. Each point was generated by averaging NQL_D across 100 task sets. The $n = 1,000$ curve lies slightly above the $n = 100$ curve, yet the

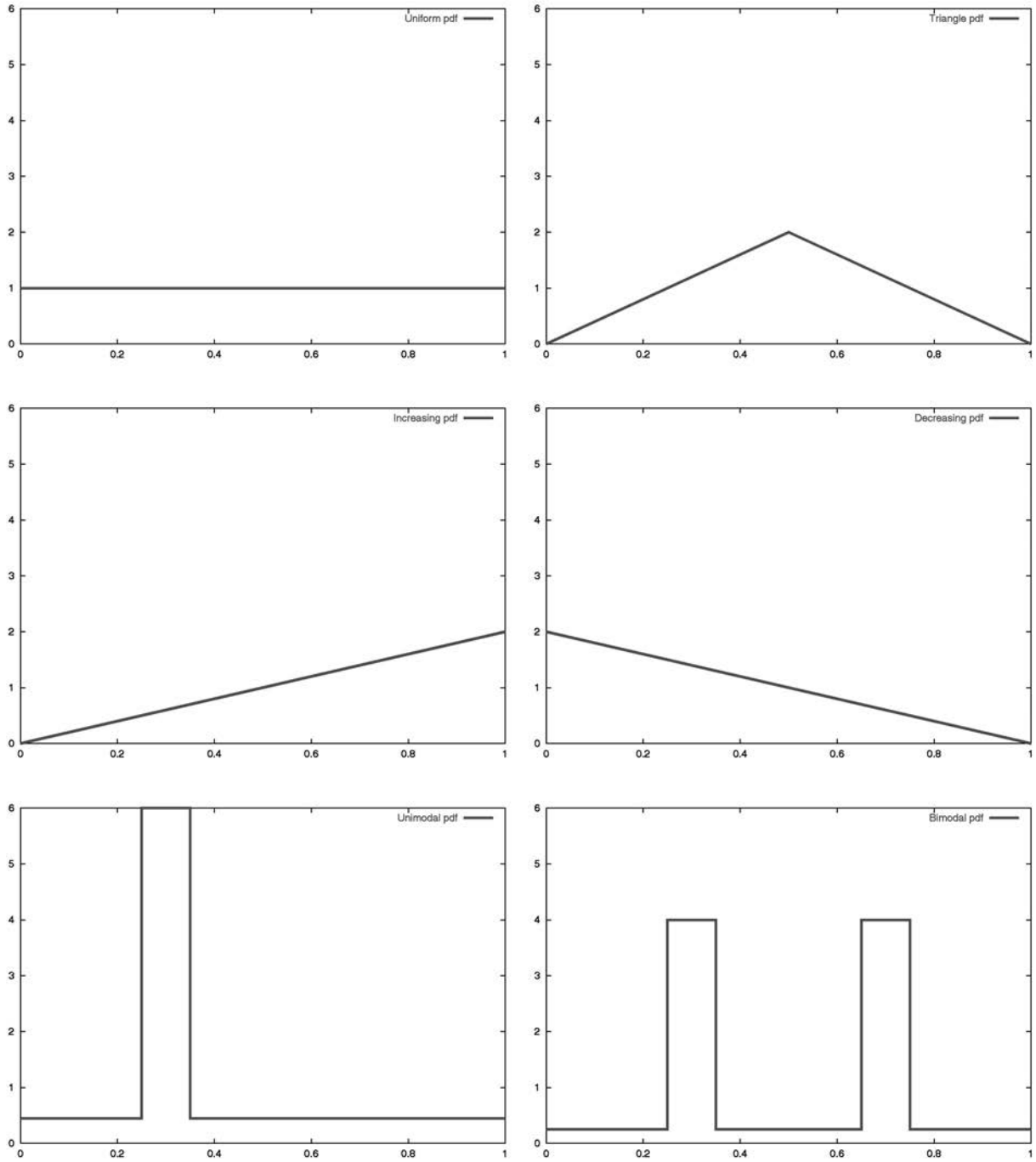

Fig. 2. Probability density functions for the six input distributions.

general shape of the curves remains the same regardless of N and input distribution: NQL_D drops immediately as l increases. In each graph, NQL_D has dropped below 1.05 at an l-value ≤ 20, for both the $n = 100$ curve and the $n = 1,000$ curve. This means that, by using only 20 (or fewer) service levels, we can adequately service task sets of 100 or even 1,000 periodic requests, dedicating no more than 5 percent processor resources beyond the amount requested. Another interpretation is that, for a fixed amount of processor resources, we can accept periodic task requests up to approximately 95 percent capacity.

Fig. 4 contains two graphs using the triangle input distribution. Fig. 4a shows NQL_D along the y-axis corresponding to each of the 100 individual task sets N for $n = 100$. Fig. 4b shows the same, for 100 task sets with $n = 1,000$. Level curves for $l = 2, 4, 6, 8, 10, 15, 20$ are shown. These graphs present the information contained within the triangle-input graph of Fig. 3 in a different way; the single point at, say, $l = 10$ in the $n = 100$ (respectively, $n = 1,000$) triangle-input graph of Fig. 3 was created from averaging the NQL_D values of the 100 points shown in the level curve for $l = 10$ in Fig. 4a (respectively, Fig. 4b). As expected, we

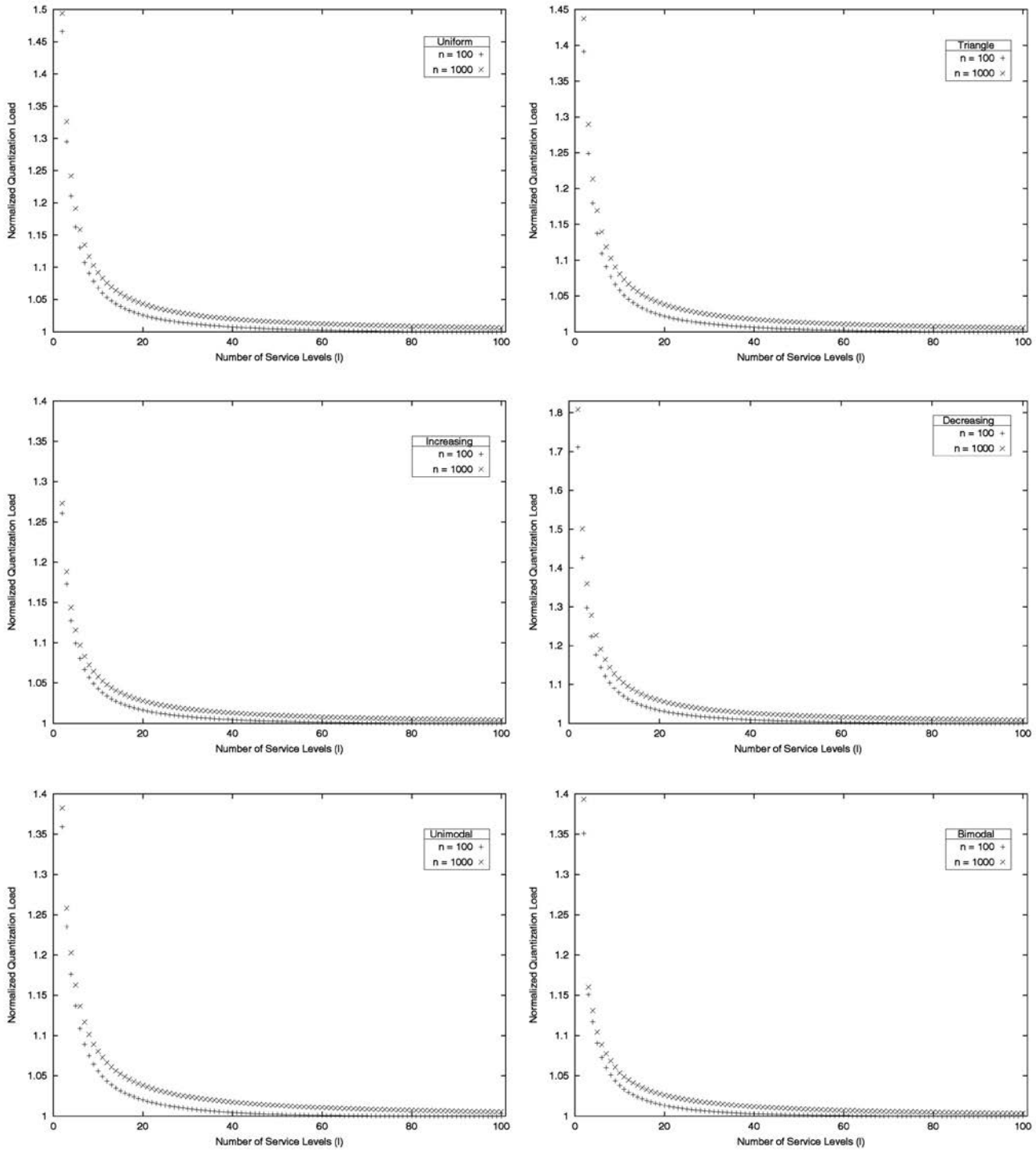

Fig. 3. Level curves in n of the ratio of quantization load to requested load versus number of service levels (l).

see that, as the number of service levels used for quantization increases, the normalized quantization load improves; that is, as l increases, NQL_D approaches 1 from above. Comparing the Fig. 4a to Fig. 4b, we can see that the variation in NQL_D decreases as the task set N increases in size from $n = 100$ to $n = 1,000$.

Figures similar to Fig. 4 for the remaining input distributions exhibit similar characteristics, and can be found in [8].

3 The Periodic Task Quantization Problem with Stochastic Input (PTQ-S)

3.1 Statement of the Problem

Let $f(\rho)$ and $F(\rho)$ be the probability density function and cumulative distribution function, respectively, representing the population of periodic tasks, with domain wholly contained within $(0, 1)$. Let μ be the mean of $f(\rho)$ and let $b \leq 1$ be the least upper bound on the domain of $f(\rho)$. A set

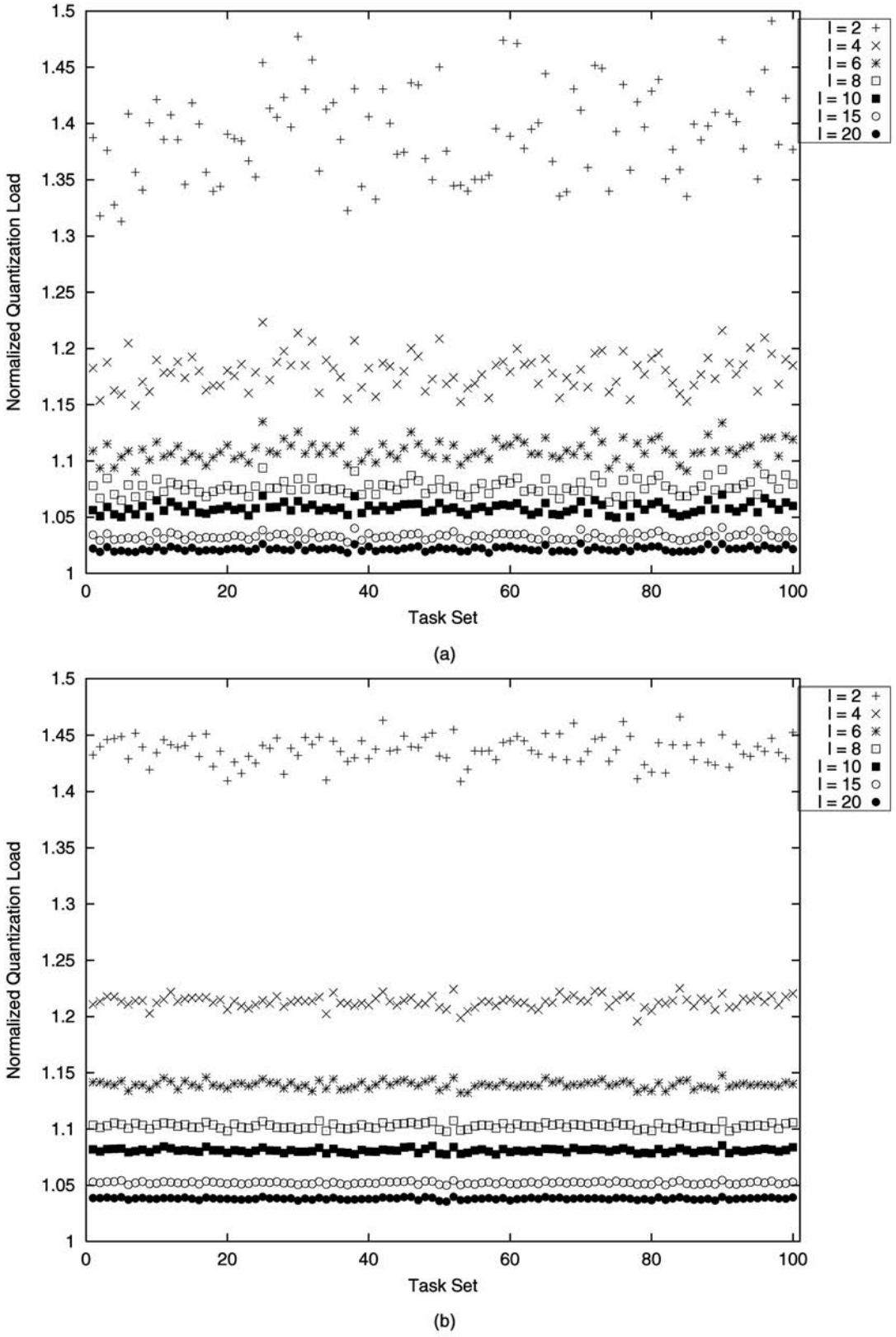

Fig. 4. Triangle input: Level curves in l of the normalized quantization load, for 100 individual task sets. (a) $n = 100$ and (b) $n = 1,000$.

$L = \{s_1, \ldots, s_l\}, 0 < s_1 < s_2 < \cdots < s_l < 1$, is a *feasible quantization set* of $f(\rho)$ if and only if $b \leq s_l$. For notational convenience, we set $s_0 = 0$. Notice that each s_j is unique. Associated with a feasible quantization set is an *implied mapping* from the domain of $f(\rho)$ into L, where $\rho \rightarrow s_j$ if and only if $s_{j-1} < \rho \leq s_j$. That is, a periodic task that requests a share of processor power equal to ρ will be able to meet its periodic deadlines if it is given a share of processor power (or

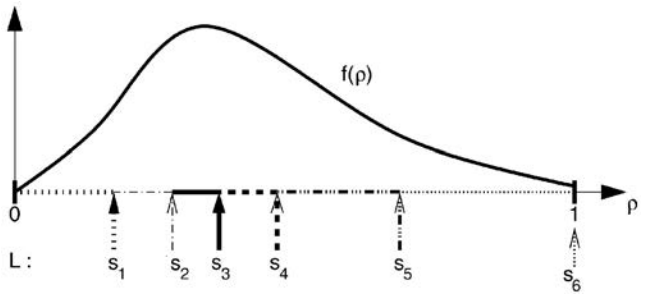

Fig. 5. Sample mapping from the domain of $f(\rho)$ to a quantization set of six service levels.

service level) equal to s_j, provided $\rho \leq s_j$. We may also write the implied mapping as $(\rho_{lower}, \rho_{upper}] \rightarrow s_j$, where $\rho_{lower} = s_{j-1}$ and $\rho_{upper} = s_j$. Fig. 5 shows a sample mapping for the PTQ-S problem.

Problem 2. *(PTQ-S) Given $f(\rho)$, $F(\rho)$, and μ as defined above, find a feasible set of l quantized service levels s_j, $j = 1, \ldots, l$, such that the following objective function is minimized:*

$$g_S(s_1, \ldots, s_l) = \sum_{j=1}^{l} \left(\int_{s_{j-1}}^{s_j} (s_j - \rho)\, f(\rho)\, d\rho \right) \tag{8}$$

$$= \sum_{j=1}^{l} \left(\int_{s_{j-1}}^{s_j} s_j\, f(\rho)\, d\rho \right) - \sum_{j=1}^{l} \left(\int_{s_{j-1}}^{s_j} \rho\, f(\rho)\, d\rho \right) \tag{9}$$

$$= \sum_{j=1}^{l} \left(s_j \int_{s_{j-1}}^{s_j} f(\rho)\, d\rho \right) - \mu \tag{10}$$

$$= q_S(s_1, \ldots, s_l) - \mu. \tag{11}$$

Notice that $g_S(s_1 \ldots s_l)$ is the *average* penalty per task of excess processor load used by the quantized set above that requested by the original task set. The second term of (11), μ, is the *average* amount of processor load requested by a task, while the first term, $q_S(s_1, \ldots, s_l)$, is the *average* quantization load, that is, the average processor load across the set of quantized tasks. In contrast, in the deterministic input case given in (3), $g_D(s_1, \ldots, s_l) = q_D(s_1, \ldots, s_l) - \rho_N$ is the *total* penalty under quantization for a particular task set N, not the average, and $q_D(s_1, \ldots, s_l)$ (respectively, ρ_N) is the total processor load for the quantized task set (respectively, for the original task set). We reach this conclusion mathematically by dividing (2) by n and taking the limit as n goes to infinity:

$$\lim_{n\to\infty} \frac{g_D(s_1, \ldots, s_l)}{n} = \lim_{n\to\infty} \left(\frac{1}{n} \left(\sum_{j=1}^{l} (n_j s_j) - \rho_N \right) \right)$$

$$= \lim_{n\to\infty} \left(\sum_{j=1}^{l} \left(\frac{n_j}{n} s_j \right) \right) - \lim_{n\to\infty} \left(\frac{\rho_N}{n} \right)$$

$$= \sum_{j=1}^{l} \left(s_j \lim_{n\to\infty} \left(\frac{n_j}{n} \right) \right) - \mu.$$

Notice that the limit of $\frac{n_j}{n}$ as n goes to infinity equals the proportion of ρ_is that fall within the interval (s_{j-1}, s_j), or $\int_{s_{j-1}}^{s_j} f(\rho)\, d\rho$. Thus, we have:

$$\lim_{n\to\infty} \frac{g_D(s_1, \ldots, s_l)}{n} = \sum_{j=1}^{l} \left(s_j \int_{s_{j-1}}^{s_j} f(\rho)\, d\rho \right) - \mu$$

$$= g_S(s_1, \ldots, s_l).$$

We can find an expression for Normalized Quantization Load for stochastic input, NQL_S, by taking the limit of (5) as n goes to infinity (notice that, in going from (12) to (13) below, we multiply the numerator and denominator by $\frac{1}{n}$):

$$NQL_S = \lim_{n\to\infty} \frac{\sum_{j=1}^{l} (n_j s_j)}{\rho_N} \tag{12}$$

$$= \lim_{n\to\infty} \frac{\sum_{j=1}^{l} \left(\frac{n_j}{n} s_j \right)}{\frac{\rho_N}{n}} \tag{13}$$

$$= \frac{\sum_{j=1}^{l} \left(s_j \lim_{n\to\infty} \left(\frac{n_j}{n} \right) \right)}{\lim_{n\to\infty} \left(\frac{\rho_N}{n} \right)} \tag{14}$$

$$= \frac{\sum_{j=1}^{l} (s_j \int_{s_{j-1}}^{s_j} f(\rho)\, d\rho)}{\mu} \tag{15}$$

$$= \frac{q_S(s_1, \ldots, s_l)}{\mu}. \tag{16}$$

Because μ is a constant for a given $f(\rho)$, both the objective function $g_S(s_1, \ldots, s_l)$ and the Normalized Quantization Load NQL_S are minimized whenever the average quantization load $q_S(s_1, \ldots, s_l)$ is minimized. The following lemma is analogous to the fact that, in the deterministic case, the largest service level in an optimal quantization set must equal the largest task density ρ_n.

Lemma 4. *Let $f(\rho)$, $F(\rho)$, and b be defined as above. Let $L_1 = \{s_1, \ldots, s_l\}$, $0 < s_1 < s_2 < \cdots < s_l < 1$, be an optimal quantization set of $f(\rho)$. Then, $s_l = b$.*

Proof. By contradiction. Suppose $s_l \neq b$. From the definition of a feasible quantization set, we know $b \leq s_l$, thus, $b < s_l$. The values currently mapped to s_l lie in the interval $(s_{l-1}, b]$. Moving s_l down to b will reduce the objective function by a nonnegligible amount equal to $(s_l - b) \int_{s_{l-1}}^{b} f(\rho)\, d\rho$. This contradicts the optimality of L_1. Thus, $s_l = b$. □

3.2 Optimal Solution through Nonlinear Programming

For a given cumulative distribution function $F(\rho)$ and given values of l and b, we can optimally solve problem PTQ-S using the method described in this section, whenever $F(\rho)$ is 1) twice differentiable and 2) not piecewise defined, over the entire domain of $F(\rho)$. In Section 3.3, we present an approximate solution for instances of PTQ-S for which $F(\rho)$ fails to have these two properties.

Rewriting $g_S(s_1, \ldots, s_l)$ from (10), we have the following optimization problem:

$$\text{Minimize } g_S(s_1, \ldots, s_l) = \sum_{j=1}^{l} \left(s_j \left(F(s_j) - F(s_{j-1}) \right) \right) - \mu$$

$$\text{subject to :} \qquad 0 < s_1 < s_2 < \ldots < s_{l-1} < s_l = b.$$

When $F(\rho)$ is twice differentiable and not piecewise defined, $f(\rho)$ and $f'(\rho)$ are also not piecewise defined. Specifically, for each of $F(\rho)$, $f(\rho)$, and $f'(\rho)$, it is possible to

write the function as a single closed form expression over its entire domain, a necessary property for applying the following method: locate a critical point of g_S and, then, verify that the point is a minimum.

To find a critical point, we set the first order partial derivatives of g_S with respect to s_j, $j = 1, \ldots, l-1$, equal to zero, yielding a set of $l-1$ simultaneous differential equations in $l-1$ unknowns. The highest service level s_l is known; from Lemma 4, we know $s_l = b$. It will then be possible to solve for each s_j, $j = 2, \ldots, l$, in terms of s_1 only. Since $s_l = b$, we can find s_1. Through back-substitution, we can then obtain the remaining values for s_j, $j = 2, \ldots, l-1$.

Taking the partial derivative of g_S with respect to s_j, $j = 1, \ldots, l-1$, we have:

$$\frac{\partial g_S}{\partial s_j} = s_j \frac{\partial F(s_j)}{\partial s_j} + \big(F(s_j) - F(s_{j-1})\big) - s_{j+1}\frac{\partial F(s_j)}{\partial s_j} \quad (17)$$

$$= (s_j - s_{j+1})\frac{\partial F(s_j)}{\partial s_j} + F(s_j) - F(s_{j-1}) \quad (18)$$

$$= (s_j - s_{j+1}) f(s_j) + F(s_j) - F(s_{j-1}). \quad (19)$$

From the equation $\frac{\partial g_S}{\partial s_j} = 0$, $j = 1, \ldots, l-1$, we can solve for s_{j+1} in terms of s_j and s_{j-1}:

$$s_{j+1} = s_j + \frac{F(s_j) - F(s_{j-1})}{f(s_j)}. \quad (20)$$

Since $s_0 = 0$, then $F(s_0) = 0$. For the equation corresponding to $\frac{\partial g_S}{\partial s_1} = 0$, we have:

$$s_2 = s_1 + \frac{F(s_1)}{f(s_1)}. \quad (21)$$

Thus, we have s_2 in terms of s_1 only. For the equation corresponding to $\frac{\partial g_S}{\partial s_2} = 0$, we have:

$$s_3 = s_2 + \frac{F(s_2) - F(s_1)}{f(s_2)}. \quad (22)$$

Using (21) to substitute for s_2 in (22) above, gives an expression for s_3 in terms of s_1 only. For the equation corresponding to $\frac{\partial g_S}{\partial s_3} = 0$, we have:

$$s_4 = s_3 + \frac{F(s_3) - F(s_2)}{f(s_3)}. \quad (235)$$

Since we already have both s_3 and s_2 in terms of s_1 only, we can use substitution to get s_4 in terms of s_1 only. In general, we can obtain an expression for s_{j+1} only in terms of s_1 after using substitution in the equation corresponding to $\frac{\partial g_S}{\partial s_j} = 0$. The final equation corresponding to $\frac{\partial g_S}{\partial s_{l-1}} = 0$ is:

$$b = s_l = s_{l-1} + \frac{F(s_{l-1}) - F(s_{l-2})}{f(s_{l-1})}.$$

After substitution, the left-hand side of this equation is the constant b, and the right-hand side is a function of s_1. Thus, we can solve for s_1. All other values of s_j, $j = 2, \ldots, l-1$, can be obtained once s_1 is known.

Notice that the feasible region, defined by $0 < s_1 < s_2 < \ldots < s_{l-1} < s_l = b$, is a convex set. If $F(\rho)$ is a convex function, then g_S is also convex, and the critical point $(s_1, s_2, \ldots, s_{l-1})$ will be a global minimum. Otherwise, the critical point $(s_1, s_2, \ldots, s_{l-1})$ is a minimum if and only if the Hessian matrix of second partial derivatives of g_S is positive definite. Since the Hessian for g_S turns out to be a symmetric tridiagonal matrix, it can be shown to be positive definite (or not) in time $O(l^2)$ [6].

Example: Solution for uniform input distribution. Due to the simplicity of the uniform distribution, namely, $f(\rho) = 1$ and $F(\rho) = \rho$, it is possible to solve for the optimal values of $s_1, \ldots, s_{l-1}$ without specifying a particular value for l. The domain of the uniform distribution is $(0, 1)$; thus, from Lemma 4, we have $s_l = 1$. Using (20), we have:

$$s_{j+1} = s_j + \frac{s_j - s_{j-1}}{1}$$
$$= 2s_j - s_{j-1}.$$

Recalling $s_0 = 0$, the first equation (corresponding to $\frac{\partial g_S}{\partial s_1} = 0$) yields: $s_2 = 2s_1$. From the second equation, we have: $s_3 = 2s_2 - s_1 = 3s_1$; from the third: $s_4 = 2s_3 - s_2 = 4s_1$; and so on, up to the $(l-1)$st equation: $s_l = ls_1$. In general, $s_j = js_1$ for $j = 2, \ldots, l$. Using the additional information that $s_l = 1,$, we have that $s_l = ls_1 = 1$. Thus, $s_1 = \frac{1}{l}$ and, for $j = 2, \ldots, l-1$, we have $s_j = \frac{j}{l}$.

3.3 An Efficient Approximate Solution

3.3.1 Algorithm Quantize-Continuous

For any given cumulative distribution function $F(\rho)$ and given values of l and b, we can find an approximate solution to problem PTQ-S using the method described here. This approximation is necessary whenever $F(\rho)$ is piecewise defined or fails to be twice differentiable; in addition, this approximation may be used whenever the complexity of $F(\rho)$ and $f(\rho)$ make the approach of Section 3.2 difficult.

In this situation, it is possible to create a discrete approximation of the pdf and use an algorithm similar to Quantize to find an estimate of the optimal quantization set L. That is, the new algorithm, called Quantize-Continuous, will find the optimal quantization set for a given approximation of a pdf. The better the pdf approximation, the closer the estimate will be to the true optimal solution for the pdf.

In particular, we can choose an integer $K > l$ and partition the interval $(0, 1)$ into K intervals $(\frac{i-1}{K}, \frac{i}{K})$, $i = 1, \ldots, K$. The right-hand endpoint of the ith interval is $e_i = \frac{i}{K}$; with e_i, we associate a discrete point mass density $m_i = \int_{\frac{i-1}{k}}^{\frac{i}{k}} f(\rho)\, d\rho$. These K ordered pairs (e_i, m_i) form the approximation of $f(\rho)$ that serves as input to the algorithm Quantize-Continuous, which selects the s_js of the optimal quantization set L from among the K endpoint values $\{e_i\}$. Fig. 6 demonstrates the approximation process for a sample pdf $f(\rho)$.

Quantize-Continuous differs from Quantize in several ways. First, the input to Quantize-Continuous is the collection of K ordered pairs (e_i, m_i), while the input to Quantize is the periodic task set N, containing n values of ρ_i. Second, the tables `Diff` and `Cumul` are replaced by the $K \times K$ tables `Sum` and `Prod`:

$$\texttt{Sum}[i][j] = \sum_{x=i}^{j} \texttt{m}[x]\ ,\ i \le j$$

$$\texttt{Prod}[i][j] = \texttt{e}[i] \cdot \texttt{Sum}[j][i]\ ,\ j \le i.$$

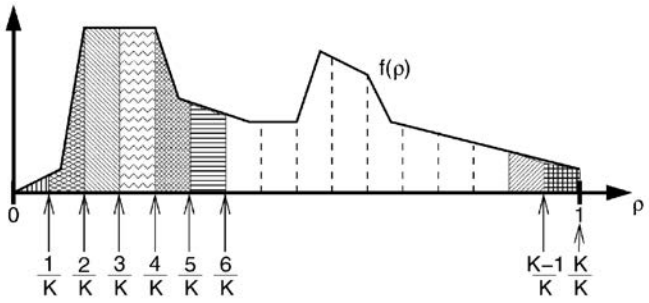

Fig. 6. Forming a pdf approximation: The area under $f(\rho)$ over an interval is paired with the right-hand endpoint of the interval.

Finally, Quantize-Continuous minimizes the average quantization load $q_S(s_1, \ldots, s_l)$ (and holds these values in a $K \times l$ table called AQL), whereas Quantize minimizes the total quantization load $q_D(s_1, \ldots, s_l)$ (and holds the values of $q_D(s_1, \ldots, s_l) - \rho_N = g_D(s_1, \ldots, s_l)$ in the Opt table). Apart from these differences, the two algorithms are very similar, in that the same code used in Quantize to build Opt (using Cumul) is exactly the same code used in Quantize-Continuous to build AQL (using Prod in the place of Cumul). Quantize-Continuous runs in time $O(K^2 l)$ and Quantize runs in time $O(n^2 l)$; these time complexities are identical since K and n simply represent the size of the input.

Note that the entry AQL $[i][j]$ holds the minimum value of q_S for a subset of the larger problem instance of PTQ-S that we wish to solve; namely, AQL $[i][j]$ is the portion of $q_S(s_1, \ldots, s_l)$ that arises from optimally choosing j service levels to quantize the first i pairs (e_1, m_1) up to (e_i, m_i). (AQL $[i][j]$ does not hold the minimum value of q_S for a smaller problem instance of PTQ-S in which the $K = i$ and $l = j$.) The pseudocode description of Quantize-Continuous can be found in [8].

3.3.2 Performance of Quantize-Continuous on Six Input Distributions

To evaluate the performance of Quantize-Continuous, we ran the algorithm on the six different input distributions described in Section 2.3.1 for a variety of values of K and l. In particular, we allowed K to take on the values $10, 15, 20, \ldots, 100$ and l the values from 2 to 50. However, in the graphs of Fig. 7, we have chosen only to display level curves of l for $l = 5, 10, 15, 20, 25, 30$. This figure contains two graphs: Fig. 7a was created using the triangle distribution as input to Quantize-Continuous and Fig. 7b using the bimodal distribution. (Graphs created using the remaining input distributions can be found in [8].) We have plotted the value of the normalized quantization load NQL_S on the y-axis corresponding to a particular value of K along the x-axis. As expected, the level curves of l approach 1 as l increases. Notice also that, for a particular value of l, as K increases, the value of NQL_S decreases slightly and immediately settles down to a particular value. For example, in Fig. 7a (triangle input), the level curve of $l = 20$ settles down to a value of $NQL_S \approx 1.045$ as early as $K = 25$. Therefore, by dividing the interval $(0, 1)$ into as few as 25 smaller intervals, we can adequately estimate the effect on processor resources due to quantization into 20 service levels.

In Fig. 7b (bimodal input), the level curves of l do not appear to settle down as quickly; instead, they possess an interesting sinusoidal shape. We, therefore, generated another set of graphs (shown in [8]) for the bimodal distribution, this time letting K take on the values $10, 15, 20, \ldots, 300$. From $K = 100$ to $K = 300$, the sinusoidal shape quickly decreases in amplitude and settles down to a particular value of NQL_S. The shape can be attributed to the endpoints of the K intervals adequately falling along the points of discontinuity of the pdf. The first peak in the bimodal distribution rises at $\rho = .25$ and falls at $\rho = .35$, and the second peak rises at $\rho = .65$ and falls at $\rho = .75$. When $K = 20, 40, 60, \ldots$, there are endpoints e_i that exactly equal .25, .35, .65, and .75; further, these K values correspond to the valleys (lower values of NQL_S) of the level curves of l.

Thus, a probability density function $f(\rho)$ with discontinuities will be better approximated (and, hence, Quantize-Continuous will perform better) whenever the K intervals are chosen such that the endpoints lie at the points of discontinuity. In fact, Quantize-Continuous does not require that the input pairs (e_i, m_i) be evenly spaced along the interval $(0, 1)$. Therefore, whenever $f(\rho)$ has many discontinuities, the endpoints e_i may be particularly chosen to fall at the points of discontinuity to achieve better performance from Quantize-Continuous.

4 Scheduling a Quantized Task Set

The algorithm PD^2 is the fastest known algorithm that will create a p-fair schedule for periodic task sets in which $\sum_{i=1}^{n} \rho_i \leq m$, where m is the number of processors [1]. Recall from Section 1.2 that each subtask has an associated slot deadline, the last eligible slot in which the subtask may be processed in a p-fair schedule. PD^2 uses these deadlines to choose subtasks for scheduling. The online implementation of PD^2 presented in [1] has the following main phases:

1. **Preprocessing**. The algorithm inserts the initial eligible subtask of each of the n tasks into a heap H, which holds all subtasks currently eligible for processing.
2. **Scheduling**. At each time slot:
   a. **Selection**. The algorithm chooses a total of m eligible subtasks to process. It chooses subtasks according to most imminent deadline and breaks ties in constant time.
   b. **Update**. For each of the m selected subtasks, the algorithm calculates the earliest time slot t at which the next subtask will become eligible. It then inserts this next subtask into a heap H_t according to its deadline. Since there are n tasks in all, the number of nonempty heaps is at most $n + 1$.

PD^2 completes the Preprocessing phase in time $O(n)$. During the Scheduling phase, PD^2 completes Selection in time $O(m \log n)$ and Update in time $O(m \log n)$. At any point in time, for each task, at most one subtask (the next one to be processed) is stored in one of several heaps: heap H if the

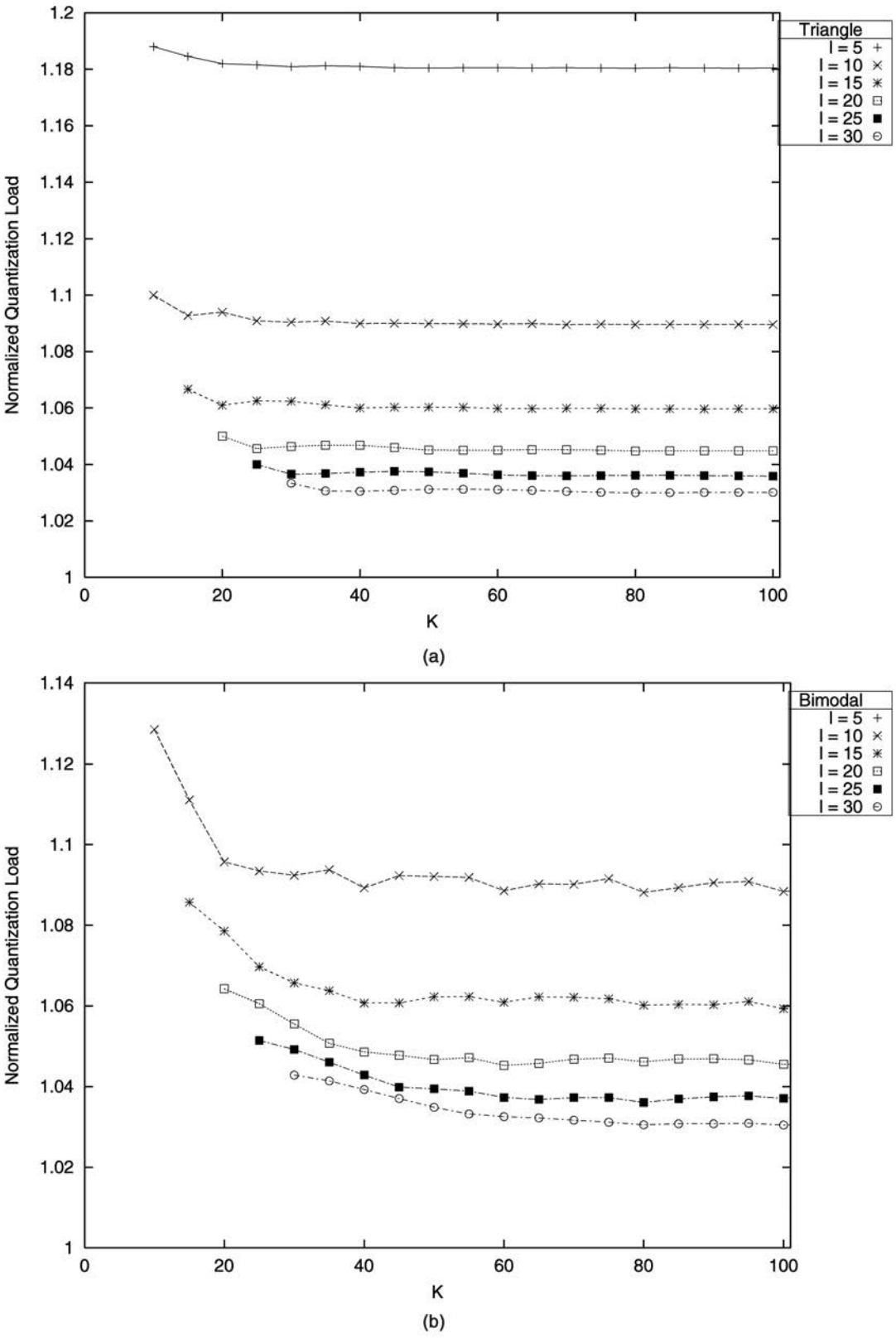

Fig. 7. Level curves in l of the normalized quantization load for $K = 10, 15, \ldots, 100$. (a) Triangle input and (b) bimodal input.

subtask is currently eligible, or heap H_t if time slot t is the next earliest slot that the subtask will be eligible.

We propose modifying PD^2 to create a new algorithm called $\mathrm{Quantized\text{-}PD}^2$ ($\mathrm{Q\text{-}PD}^2$). We use the same priority definition as PD^2; that is, we use the same rules during the Selection phase to choose m eligible subtasks for processing. Taking advantage of the quantized input, we replace the collection of heaps with a set of l queues, one for each service level s_j. During the Preprocessing phase, the initial eligible subtask of each of the n tasks is inserted in arbitrary

order into the queue corresponding to its assigned service level. Initially, all subtasks within a given queue have the same deadline and, hence, the same priority. Choosing the eligible head-of-line subtask with the highest priority from among the l queues can be done in time $O(1)$ (recall that l is a constant; it does not depend on n or m). The Selection phase can, therefore, be completed in time $O(m)$. Next, for each of the m selected subtasks, the Update phase involves: 1) calculating the time `t_next` at which the task's next subtask will become eligible, 2) calculating the priority of the next subtask at time `t_next`, and 3) placing the next subtask at the end of its queue. Each of these three actions for updating a single task requires time O(1) so, in total, the Update phase requires $O(m)$.

Therefore, Q-PD^2 has a per slot time complexity of $O(m)$ as compared to $O(m \log n)$ for PD^2. The Preprocessing phase time complexity remains unchanged at O(n). The pseudocode description of Q-PD^2 is given in the appendix.

5 Conclusion

We have attempted to simplify the periodic tasks scheduling problem by making a trade off between processor load and computational complexity. In particular, we sought to quantize processor power by determining a set of service levels that would strike a balance between the two conflicting goals of simplicity and performance. We addressed the issue of determining this set of service levels given 1) a fixed set of task requests (Periodic Task Quantization Problem with Deterministic input), and 2) the probability density function of task requests (Periodic Task Quantization Problem with Stochastic input), giving optimal solutions in each case. Finally, we have shown that the scheduling of a set of periodic tasks is greatly simplified through quantization and have presented a fast online algorithm that schedules quantized periodic tasks.

Appendix

The Q-PD^2 Algorithm for Scheduling a Quantized Task Set

```
Q = BuildQueues(rho) ;    // Preprocessing phase t = 0
t = 0;
while (true)          // Start of Scheduling phase
    repeat {
        T = ExtractMin(Q) ;
        Schedule task T in slot t ;
        t_next = the earliest future time at which
        T will be eligible again ;
        T.nextEligible = t_next;
        T.priority = Determine T's priority at time
        t_next ;
        Enqueue(Q, T) ;
    }
    until m tasks have been scheduled in slot t ;
    t = t + 1 ;
}
```

References

[1] J.H. Anderson and A. Srinivasan, "A New Look at Pfair Priorities," technical report, Univ. of North Carolina, Sept. 1999.

[2] J.H. Anderson and A. Srinivasan, "Mixed Pfair/ERfair Scheduling of Asynchronous Periodic Tasks," *J. Computer and System Sciences,* 2001.

[3] S.K. Baruah, N.K. Cohen, C.G. Plaxton, and D.A. Varvel, "Proportionate Progress: A Notion of Fairness in Resource Allocation," *Algorithmica,* vol. 15, no. 6, pp. 600-625, 1996.

[4] S.K. Baruah, J.E. Gehrke, and C.G. Plaxton, "Fast Scheduling of Periodic Tasks on Multiple Resources," *Proc. Ninth Int'l Parallel Processing Symp.*, pp. 280-288, Apr. 1995.

[5] M.L. Dertouzos and A.K.-L. Mok, "Multiprocessor On-Line Scheduling of Hard-Real-Time Tasks," *IEEE Trans. Software Eng.,* vol. 15, no. 12, pp. 1497-1506, Dec. 1989.

[6] I. Dhillon, "A New Algorithm for the Symmetric Tridiagonal Eigenvalue-Eigenvector Problem," PhD thesis, Univ. of California, Berkeley, 1997.

[7] R. Hassin and A. Tamir, "Improved Complexity Bounds for Location Problems on the Real Line," *Operations Research Letters,* vol. 10, pp. 395-402, 1991.

[8] L.E. Jackson and G.N. Rouskas, "Optimal Quantization of Periodic Task Requests on Multiple Identical Processors," technical report, North Carolina State Univ., Jan. 2002.

[9] C. Lea and A. Alyatama, "Bandwidth Quantization and States Reduction in the Broadband ISDN," *IEEE/ACM Trans. Networking,* vol. 3, pp. 352-360, June 1995.

Laura E. Jackson received the BS degree in math in 1995, and the MS degree in operations research in 1997, from the College of William and Mary, Williamsburg, Virginia. She is currently pursuing the PhD degree in computer science at North Carolina State University, with a research focus on real-time packet scheduling and QoS in all-optical networks. Since April 2000, she has worked in Advanced Network Research at MCNC Research and Development Institute, working on traffic scheduling and protocol design for an all-optical broadcast LAN. She is a student member of the IEEE and the IEEE Computer Society.

George N. Rouskas (S '92, M '95, SM '01) received the diploma in electrical engineering from the National Technical University Athens (NTUA), Athens, Greece, in 1989, and the MS and PhD degrees in computer science from the College of Computing, Georgia Institute of Technology, Atlanta, Georgia, in 1991 and 1994, respectively. He is currently a professor with the Department of Computer Science, North Carolina State University, which he joined in August 1994. During the 2000-2001 academic year, he spent a sabbatical term at Vitesse Semiconductor, Morrisville, North Carolina, and in June 2000 and December 2002, he was an invited professor at the University of Evry, France. His research interests include network architectures and protocols, optical networks, multicast communication, and performance evaluation. He is a recipient of a 1997 US National Science Foundation Faculty Early Career Development (CAREER) Award, and a coauthor of a paper that received the Best Paper Award at the 1998 SPIE Conference on All-Optical Networking. He also received the 1995 Outstanding New Teacher Award from the Department of Computer Science, North Carolina State University, and the 1994 Graduate Research Assistant Award from the College of Computing, Georgia Tech. He was a coguest editor for the *IEEE Journal on Selected Areas in Communications*, special issue on protocols and architectures for next generation optical WDM networks, published in October, 2000, and is on the editorial boards of the *IEEE/ACM Transactions on Networking*, *Computer Networks*, and *Optical Networks*. He is a senior member of the IEEE and IEEE Computer Society, and a member of the ACM and of the Technical Chamber of Greece.