To appear in 1995 IEEE Real-Time Technology and Applications Symposium, Chicago, Illinois, May 1995

Probabilistic Performance Guarantee for Real-Time Tasks with Varying Computation Times*

T.-S. Tia Z. Deng M. Shankar M. Storch J. Sun L.-C. Wu J. W.-S. Liu
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Abstract

This paper describes how the scheduling algorithms and schedulability analysis methods developed for periodic tasks can be extended to provide performance guarantees to semi-periodic tasks. Like periodic tasks, the requests in a semi-periodic task are released regularly. However, their computation times vary widely. We focus on systems where the total maximum utilization of the tasks on each processor is larger than one. Hence according to the existing schedulability conditions for periodic tasks, we cannot guarantee that the semi-periodic tasks are schedulable, even though their total average utilization is very small. We describe two methods of providing probabilistic schedulability guarantees to the semi-periodic tasks: The first method, called probabilistic time-demand analysis, is a modification of the exact schedulability test for periodic tasks. The second method, called the transform-task method, transforms each task into a periodic task followed by a sporadic task. The transform-task method can provide an absolute guarantee to requests with shorter computation times and a probabilistic guarantee to the longer requests.

1 Introduction

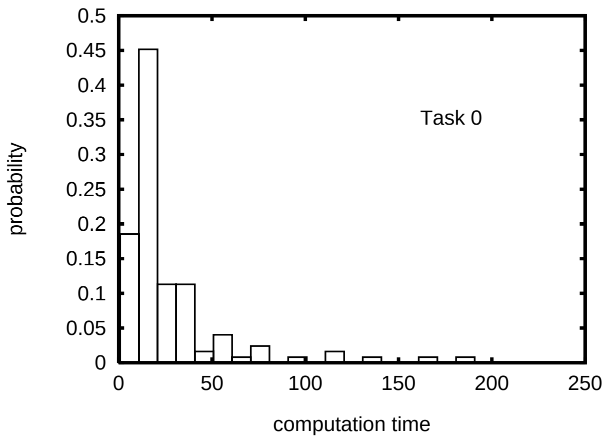
The periodic-task model [9] has been used extensively to characterize real-time applications. According to this model, computations that provide continuous control, signal processing, and monitoring functions are modeled as periodic tasks. Each periodic task is an infinite stream of requests for computation. The parameters of the task are its period, computation time and (relative) deadline. The *period* of a periodic task is the minimum length of the time intervals between arrivals of consecutive requests. Its *computation time* is the maximum computation time of the individual requests. Its *deadline* is the maximum allowable response time of each request. These parameters are known *a priori.*

A *schedulability condition* of a scheduling algorithm is a condition on the parameters of a system that is scheduled according to the algorithm; all requests in the system complete in time whenever the condition is met. Schedulability conditions exist for many well-known algorithms for scheduling periodic tasks, including the rate-monotonic (RM) algorithm, the deadline-monotonic (DM) algorithm, and the earliest-deadline-first (EDF) algorithm [6, 7]. These conditions are conservative but not overly pessimistic. For applications that fit the periodic task model, the schedulability conditions can serve as design rules. By ensuring that the chosen task parameters meet the schedulability condition(s) of the scheduling algorithm used in the system, the designer is ensured of the timing correctness of the system.

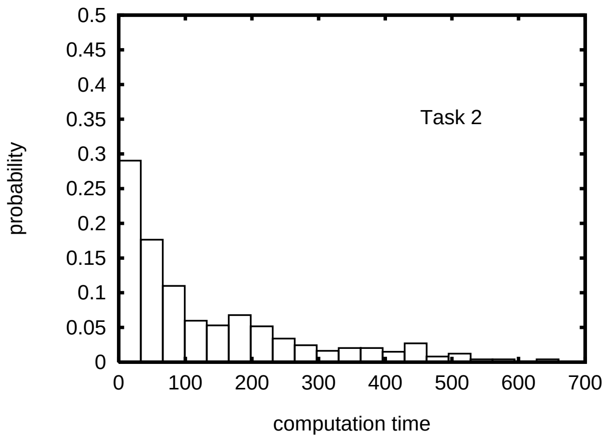
We focus here on a class of applications that do not fit the periodic task model. Although requests for computation in these applications are released periodically, their computation times vary widely. This variation is sometimes due to the difference in the amounts of input data processed in different periods. Also, different requests may execute different conditional branches that vary widely in complexity. For example, Figure 1 gives the computation-time histograms of two such tasks. The maximum computation time of the requests in each task is significantly larger than the average computation time of the task. If we were to model these tasks according to the periodic task model, the utilization of each task (i.e., the ratio of maximum computation time to period) may be larger than one. According to the existing schedulability conditions, we cannot guarantee that such a task is schedulable, even though its average utilization is very small.

Hereafter, we call tasks exemplified by the tasks in Figure 1 *semi-periodic* tasks. Requests in such a task are released periodically, but the computation time of each request is a random variable. Like other parameters of the task, the probability density function of the computation times of requests in it is known *a priori.* As a specific example, we consider an actual application system that contains 30 semi-periodic tasks, including the tasks shown in Figure 1. Detailed timing information about the system can be found in [8]. The tasks are assigned and bound to three processors so that the maximum utilizations of the processors, using the maximum computation times of the tasks, are as equal as possible. The assignments of tasks are listed in Appendix A. The maximum utilizations of the processors are 145%, 147%, and 145%, respectively.

*This research is supported in part by ONR under contract N00014-92-J-1815.



(a)



(b)

Figure 1: The Computation-Time Histograms of Two Tasks.

However, the processors are very much under-utilized on the average; the average utilizations of the three processors are 25%, 13%, and 15%, respectively.

A natural question arises as to whether it is possible to provide some satisfactory form of schedulability guarantee for systems where the maximum utilization of each processor is more than 100%. This paper extends the existing schedulability-analysis methods for periodic tasks [3-5,7] to provide probabilistic schedulability guarantees for semi-periodic tasks. Using a probabilistic performance guarantee, the designers can make the appropriate tradeoffs between using fewer processors to execute the tasks versus having a higher degree of confidence that all requests will meet their deadlines.

We describe two methods of providing schedulability guarantee for semi-periodic tasks. The first method assumes that the semi-periodic tasks are scheduled according to a fixed-priority algorithm. This method, called the *probabilistic time-demand analysis* method, is a modification of the exact schedulability test [5] for periodic tasks. The modification takes into account the variations in the computation times and is similar to the approach taken in [2]. It provides the probability that requests of each task will meet their deadlines. The second method, called the *transform-task* method, transforms each semi-periodic task into a periodic task followed by a sporadic task. The periodic tasks are schedulable according to known schedulability conditions. Two different algorithms are used to schedule the periodic and sporadic tasks. The first algorithm uses sporadic servers [12] to execute the sporadic tasks while the periodic tasks are scheduled on a fixed-priority basis. The second algorithm uses the EDF algorithm to schedule the periodic tasks and a slack stealing algorithm [1, 14] to schedule the sporadic tasks. The probabilistic time-demand analysis is applicable only to tasks with deadlines equal to or less than their periods while the transform-task method does not have this restriction.

The remainder of this paper is organized as follows: In Section 2, we introduce the notations needed in later sections. In Section 3, we present the probabilistic time-demand analysis method. Section 4 describes the transform-task method. Simulation results for the two approaches to scheduling the semi-periodic tasks are also provided. Section 5 concludes the paper and discusses future work.

2 Problem Formulation

The system we consider consists of a set of N independent, semi-periodic tasks to be scheduled on a single processor. (For multiprocessor systems, we assume that the tasks have been assigned to the processors so we can analyze each processor individually.) The parameters of each semi-periodic task include its *period* p_i and relative hard *deadline* d_i. Let $T_{i,j}$ denote the j-th request of task T_i. The computation time of $T_{i,j}$ is $c_{i,j}$. For a given i, $c_{i,j}$, for $j = 1, 2, \cdots$, are identically distributed according to the probability density function $f_i(t)$. Again, for all tasks T_i, the parameters p_i, d_i, and $f_i(t)$ are known before the system begins execution. Furthermore, we assume that the computation time $c_{i,j}$ of each request $T_{i,j}$ becomes known when $T_{i,j}$ is released. We let $r_{i,j}$ be the *release time* of $T_{i,j}$ and $d_{i,j} = r_{i,j} + d_i$ be its absolute hard deadline.

The (average) *utilization* of a task T_i, denoted by u_i, is the average fraction of available processor time that T_i uses. u_i is equal to $E[c_{i,j}]/p_i$, where $E[y]$ denotes the expected value of the random variable y. The total average utilization of the system, denoted by U, is equal to $\sum_{i=1}^{N} u_i$. Let $\hat{c}_i$ denote the maximum computation time of all requests in T_i. The total maximum utilization of the system is $\sum_{i=1}^{N} \hat{c}_i/p_i$. We assume that the total maximum utilization is larger than 1 and the total average utilization is less than 1.

According to the transform-task method, each semi-periodic task T_i is divided into a periodic task P_i and a sporadic task S_i. The periodic task P_i has period p_i and computation time c_i. c_i is a parameter chosen by the designer; it is in the range $(0, \hat{c}_i)$ as shown in Figure 2. The j-th request $P_{i,j}$ of P_i is

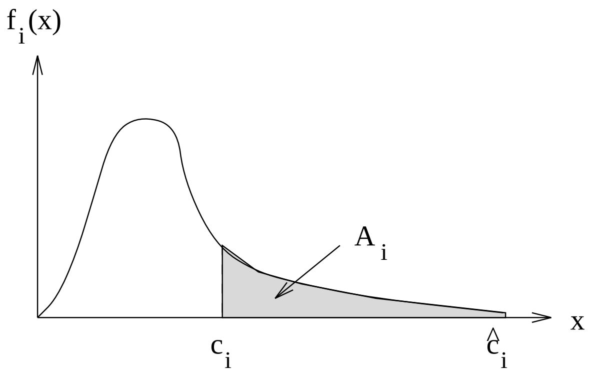


Figure 2: The probability density function of task T_i.

released at $r_{i,j}$, the release time of the j-th request of T_i. If the computation time $c_{i,j}$ of $T_{i,j}$ is larger than c_i, a request of S_i arrives and is ready for execution at the time when c_i units of $T_{i,j}$ completes. Hence the probability A_i that a sporadic request in S_i arrives in each period of T_i is equal to the area of the shaded region under the probability density function $f_i(t)$, as shown in Figure 2. We always choose c_i, and hence A_i, so that all the periodic tasks $P_i, 1 \leq i \leq N$, are schedulable.

3 Probabilistic Time-Demand Analysis

Again, one way to schedule semi-periodic tasks is to treat them as if they were periodic tasks and schedule them on a fixed-priority basis. (In this section, we index the tasks so that the priority of T_i is higher than that of T_j if $i < j$.) Because the schedulability test based on the maximum computation times fails, we seek to find the probability that any request $T_{i,j}$ in T_i will meet its deadline. Our method for finding this probability is a straightforward extension of the time-demand analysis method used to derive the exact schedulability test [5]. Again, our method assumes that the relative deadline d_k is less than or equal to the period p_k for every task T_k.

To determine the probability that a request $T_{i,j}$ will meet its deadline, we first bound the total amount of processor time $w_i(t)$ demanded by all the higher priority tasks and the request $T_{i,j}$ since the release of $T_{i,j}$. At most $R_k(t) = \lceil t/p_k \rceil$, $k = 1, 2, \cdots, i-1$, requests of any higher priority task T_k can be released in the interval $[r_{i,j}, r_{i,j} + t)$ for any $t > 0$. Without loss of generality, we call these requests $T_{k,1}, T_{k,2}, \cdots, T_{k,R_k(t)}$. An upper bound of $w_i(t)$ in terms of the computation times of these requests is:

$$w_i(t) \leq c_{i,j} + \sum_{k=1}^{i-1} \sum_{l=1}^{R_k(t)} c_{k,l} \tag{1}$$

The request $T_{i,j}$ can meet its deadline if $w_i(t)$ becomes equal to or less than t at some time instant $r_{i,j} + t$ before or at the deadline $r_{i,j} + d_i$ of $T_{i,j}$.

Let $\mathbf{W}_i(t, x)$ be the (cumulative) probability distribution of $w_i(t)$, i.e., the probability of $w_i(t) \leq x$. The probability M_i that $T_{i,j}$ can meet its deadline is at least equal to $\mathbf{W}_i(d_i, d_i)$. A better lower bound of M_i is

$$M_i \geq \max_{t \in E} \mathbf{W}_i(t, t) \tag{2}$$

where E is the set of time instants containing d_i, and $p_k, 2p_k, \cdots, \lfloor d_i/p_k \rfloor p_k$, for $k = 1, 2, \cdots, i-1$.

A practical problem is how to compute the probability distribution of $\mathbf{W}_i(t, x)$ from the given probability density functions $f_k(x)$, $k = 1, 2, \cdots, i$. The algorithm described by the pseudo code in Figure 3 uses the bound (1) and assumes the statistical independence of the computation times of all requests. If the number of terms X in (1) is small (equal to or less than 10 in the pseudo code), the probability density function $\mathbf{w}_i(t, x)$ of $w_i(t)$ is computed by convolving the probability density functions of the random variables in the sum. Although this method is computationally expensive, we only have to do a few convolutions since X is small. When the number X becomes large, the central limit theorem applies. The probability distribution of $w_i(t)$ can be approximated by a normal distribution whose mean and variance are the sums of the means and variances of all the random variables in the sum, respectively.

In our 3-processor example mentioned earlier, the total maximum utilization of the processors are 145%, 147%, and 145%, respectively. To apply the probabilistic time-demand analysis on the system, we let the relative deadline of every task be equal to its period. We used the empirical computation-time histograms of the 30 tasks as estimates of their probability density functions. Assuming that the computation times of all requests are statistically independent, we found that all but two of the tasks meet their deadlines with probability one. The probabilities of meeting deadlines for the other two tasks are 0.9964 and 0.9928, respectively. We will return in Section 4 to discuss the accuracy of this prediction.

Unfortunately, the computation times of individual requests are not statistically independent. In the system studied here, the computation times of requests in each task are correlated with that of requests in many other tasks, although in most cases only weakly. The algorithm in Figure 3 underestimates the variation of $w_i(t)$. As a consequence, the probability of meeting deadlines thus computed may be overly optimistic. Given the covariance of computation times of different requests, we can compute the probability distribution $\mathbf{W}_i(t, x)$ more accurately. However, the computation is more expensive, especially when the

Input : Task set $\mathbf{T} = \{T_i : 1 \leq i \leq N\}$ and the probability density functions $f_i(x)$ $i = 1, 2, \cdots, N$.

Output : The probability M_i that each task T_i will meet its deadline, computed according to (2).

Algorithm :

```
Define MAX_TERMS 10
For each $T_i, 1 \leq i \leq N$ Do
    $M_i = 0$
    For each $T_k, 1 \leq k < i$ Do
        $M_{i,k} = 0$
        For each $j, 1 \leq j \leq \lfloor d_i/p_j \rfloor$ Do
            $t = j \cdot p_k$
            $X = \sum_{l=1}^{i-1} \lceil t/p_l \rceil$
            If $X <$MAX_TERMS Then
                Compute $\mathbf{w}_i(t,x)$ by convolving the
                    probability density functions of
                    the terms in (1)
                Compute $\mathbf{W}_i(t,x)$ from $\mathbf{w}_i(t,x)$
            Else
                Compute $\mathbf{W}_i(t,x)$ by approximating it
                    by a normal distribution $\mathcal{N}$
            $M_{i,k} = \max\{M_{i,k}, \mathbf{W}_i(t,t)\}$
        EndFor
        $M_i = \max\{M_i, M_{i,k}\}$
    EndFor
EndFor
```

Figure 3: Pseudo Code of the Probabilistic Time Demand Analysis.

number of terms is small and even the meta-form of the central limit theorem [3] cannot be applied.

The bound given by (1) implicitly assumes that the request $T_{i,j}$ being analyzed is released at a critical instant [9]. Consequently, the sum in the right-hand side of (1) gives an upper bound of the processor-time demand of $T_{i,j}$ and all higher priority tasks only when the deadline and maximum computation time of every task are less than its period. Otherwise, some higher priority requests that are released earlier than $T_{i,j}$ may not have completed at $r_{i,j}$ and also demand processor time during the interval $[r_{i,j}, r_{i,j}+d_i)$. We can account for this extra time demand and improve the reliability of the probabilistic time-demand analysis by using the following upper bound:

$$w_i(t) \leq c_i + \sum_{k=1}^{i-1} \sum_{l=1}^{R_k(t)} c_{k,l} + \sum_{k=1}^{i-1} \sum_{l=1}^{\infty} (1 - M_k)^l c_{k,-l} \quad (3)$$

where $c_{k,-l}$ denotes the computation time of the request of T_k that was released l periods before $r_{i,j}$. When every task has a high probability of meeting its deadlines (i.e., $M_k \approx 1$), this bound is acceptably tight.

4 Task Transformation

We now present the transform-task method, which can be used to provide an absolute guarantee for requests with short computation times and a probabilistic guarantee for the other requests. As stated in Section 2, we transform each semi-periodic task T_i into a periodic task P_i followed by a sporadic task S_i.

Characteristics of the Transformed Tasks

The periodic task P_i of a semi-periodic task T_i has the same period p_i as T_i. The maximum computation time of P_i is c_i. c_i is used as the computation time of P_i when we analyze the schedulability of all N periodic tasks. The relative deadline of P_i is equal to $\alpha_i d_i$ for some α_i in the range $(0, 1)$. α_i and c_i are chosen so that all requests in P_i for all $i = 1, 2, \cdots, N$, surely meet their deadlines.

Again, a request $S_{i,j}$ in the sporadic task S_i of T_i arrives in the j-th period only if the computation time of the request $T_{i,j}$ is larger than c_i. This sporadic request arrives when the corresponding request $P_{i,j}$ in P_i completes. Its computation time is distributed according to the probability density function

$$g_i(x) = \begin{cases} 0 & \text{for } x < 0 \\ f_i(x + c_i)/A_i & \text{for } x \geq 0 \end{cases} \quad (4)$$

The probability of arrival A_i of a sporadic request in the j-th period of T_i is equal to the probability that $c_{i,j} > c_i$. The relative deadline of sporadic requests in S_i is $(1 - \alpha_i)d_i$. We can only provide a probabilistic guarantee to sporadic requests. With a slight abuse of the term, we also refer to the "period" of the sporadic task S_i as p_i.

In our subsequent discussion, we assume that α_i and A_i are equal to α and A, respectively, for all $i = 1, 2, \cdots, N$ except when stated otherwise. This assumption is made here solely for the sake of simplifying our presentation. In practice, we may want to choose different A_i and α_i for different semi-periodic tasks depending on their importance. For example, one strategy is: the more important a task, the smaller the probability A_i that any request in it might have a sporadic portion (whose completion by its deadline is not absolutely guaranteed). Similarly, we can also choose differing α_i's, which gives us another dimension of tradeoff. It is impractical to tune the $2N$ parameters α_i's and A_i's manually. However, with the help of a design tool, such as the PERTS schedulability analyzer [10], the designer can adjust these parameters one or several at a time and use the tool to find a good tradeoff. The use of PERTS for this purpose is described in [8].

Using RM and Sporadic Servers

One way to schedule the transformed tasks is to schedule the periodic tasks on a fixed-priority basis and use one or more sporadic servers to schedule the sporadic requests. The sporadic server scheme [12] is a bandwidth preserving scheme that has good performance. According to this scheme, a sporadic server may serve one or more client sporadic tasks. For each server, there is a FIFO queue containing all the released requests of the client sporadic tasks that are served by the server. The request at the head of this queue executes whenever the server is scheduled by the operating system.

We have examined the performance of the sporadic server scheme when used together with the RM algorithm to schedule the periodic tasks and sporadic tasks transformed from the 30 semi-periodic tasks in our 3-processor example. Tables B1 and B2 in Appendix B summarize the average response times obtained by simulating the system using PERTS when the client sporadic tasks are served by the sporadic servers described in Appendix A. (The lengths of the 99% confidence intervals are negligibly small.) The probability of sporadic task arrival A was tried for $A = 30\%$ and $A = 20\%$ for all tasks. In this experiment, the deadline of each periodic task is equal to the period of the corresponding semi-periodic task. The parameters of the sporadic servers were chosen by the PERTS schedulability analyzer so that all the periodic tasks and the servers (treated as periodic tasks) are schedulable according to the RM algorithm. In particular, the period of each server is slightly smaller than the shortest period of all its client tasks. Therefore the server has a higher priority than the corresponding periodic tasks of all of its clients. During simulation, the tool reports that a sporadic request misses its deadline whenever the request does not complete within one period of the sporadic server serving it. This choice of relative deadlines of sporadic tasks implies different α_i's for different tasks. We will show later in this section that by making the relative deadlines of the sporadic tasks dependent on the periods of the sporadic servers serving them, it becomes easier to analytically compute the probability of any sporadic task missing its deadlines.

The data summarized in Appendix B shows that tasks 2 and 16 miss their deadlines. The probabilities of these occurrences are 0.0095 and 0.0204 when $A = 30\%$ and are 0.0076 and 0.0139 when $A = 20\%$. Compared with the corresponding probabilities obtained from the probabilistic time-demand analysis method described in Section 3 (which are 0.0036 and 0.0072, respectively), these probabilities are larger despite the fact that the relative deadlines are larger. We suspect that the more optimistic conclusion derived from the probabilistic time-demand analysis is due to the errors introduced by (a) the assumption of statistical independence of computation times and (b) the negligence of the extra time demanded by the yet to be completed higher priority requests that are released earlier than the request being analyzed, i.e., the last term on the right-hand side of (3).

Analytical Model

We now present an analytical model based on which we can compute an upper bound of the probability that a sporadic task served by a sporadic server will miss its deadline. This model makes the following assumptions about the parameters of the sporadic servers and the clients served by each server.

1. The sporadic tasks are clustered according to the periods of their corresponding semi-periodic tasks. A cluster of sporadic tasks with period p is served by a server of the same period p.
2. The sporadic requests are served on a FIFO basis by the server.
3. Each server, together with all other servers whose parameters are chosen in the same manner and all the periodic tasks, is schedulable according to the exact schedulability test [5].

In practice, the periods of the semi-periodic tasks may be distinct, as in the case of our example application system. Because the total maximum utilization of the processor is larger than one, we cannot have a sporadic server for each sporadic task. In this case, we cluster tasks with similar periods together and use one sporadic server for each cluster. The period of the server should be the minimum of the periods of the tasks.

We consider each server that has period p and n clients. Let D_i and C_i denote the relative deadline and maximum computation time of sporadic task S_i, respectively. Let $C = \max\{C_i : 1 \leq i \leq n\}$ and $K = \min\{\lceil D_i/p \rceil : 1 \leq i \leq n\}$. The execution time budget of the server is chosen to be C/K. In other words, a sporadic request $S_{i,j}$ that arrives when the queue of its server is empty can surely complete within K server periods, i.e., D_i units of time. A sporadic request is said to miss its deadline if it is not completed within K server periods after its arrival. The probability of missed deadlines computed based on this definition is an upper bound of the actual probability of missed deadlines.

Let h_x denote the probability that a request $S_{i,j}$ in a sporadic task S_i requires x server periods to complete. Let b_m denote the probability that the backlog in the server queue when $S_{i,j}$ arrives requires m periods to complete. The probability that a sporadic task S_i will miss its deadline is given by

$$\Pr[S_i \text{ misses its deadline}] = \sum_{x=1}^{K} h_x \sum_{m=K-x+1}^{\infty} b_m. \quad (5)$$

We let q_y denote the probability that the number of server periods δ required to complete all the sporadic tasks arriving during one server period is equal to y. $H(z)$ and $Q(z)$ are the generating functions of $\{h_x\}$ and $\{q_y\}$, respectively. When the computation times of different semi-periodic tasks are statistically independent, we have $Q(z) = (H(z))^n$. In the simplest case, $K = 1$. (This is the choice made in Appendix A.) $H(z) = 1 - A + Az$. For other choices of K, $H(z)$ can be derived easily from the probability density function $f_i(x)$ of each task T_i.

To find b_m, we note that for $m \geq 1$,

$$b_m = \sum_{y=0}^{nK} q_y b_{m-y+1} \quad (6)$$

From the definition of b_m, $b_m = 0$ for $m < 0$. Hence we only need to take care of the boundary condition when $m = 0$ as follows:

$$(1 - q_0 - q_1)b_0 = q_0 b_1 \tag{7}$$

From Eqs.(6) and (7), we find that the generating function $B(z) = \sum_{m=0}^{\infty} b_m z^m$ is given by

$$B(z) = \frac{(1 - E[\delta])(1 - z)}{Q(z) - z} \tag{8}$$

when $E[\delta] < 1$, i.e., the expected number of server periods required to complete all the sporadic requests arriving in any period is less than 1. The stationary distribution $\{b_m\}$ does not exist when $E[\delta] \geq 1$.

For all choices of K, $(1 - b_0)$ gives the probability that a sporadic request arrives to a non-empty server queue. Consequently, $A(1 - b_0)$ gives an upper bound for the probability that any sporadic task misses its deadline. We can, of course, get a tighter upper bound by evaluating the value of the sum in Eq. (5). In the special case of $K = 1$, $q_0 = (1 - A)^n$, $E(\delta) = nA \quad (< 1)$, and $b_0 = (1 - nA)/(1 - A)^n$. (If different tasks have different probability of arrival A_i, $b_0 = (1 - \sum_{i=1}^{n} A_i)/\prod_{i=1}^{n}(1 - A_i)$.)

This method, like the probabilistic time-demand analysis method, assumes the statistical independence of the computation times of different semi-periodic tasks. Consequently, the upper bound thus derived may also be overly optimistic when the computation times are not statistically independent. One way to improve the analytical model is to make h_x, the probability of a sporadic request requiring x server periods to complete, dependent on the amount of backlog at the time of its arrival. An example is to make h_x larger for larger x when the backlog is large. The practical challenge of this approach is how to characterize this dependency so that the model is as accurate as possible while remaining tractable at the same time. We are examining the detailed execution traces of the tasks in our example in order to validate several possible dependency relations between h_x and the server backlog.

Using EDF and Slack Stealer

We can also schedule the semi-periodic tasks according to the EDF algorithm. This algorithm, being optimal, allows the tasks to meet their deadlines whenever it is possible to do so. However, the EDF algorithm is known to be unstable during transient overloads. For this reason, rather than applying the EDF algorithm directly to semi-periodic tasks, we again transform each task into a periodic task and a sporadic task. The periodic tasks are scheduled on the EDF basis. The sporadic requests of all tasks are then scheduled according to the slack stealing algorithm described in [14]. This algorithm differs from the slack stealing algorithms described in [4, 11, 13], which assume that periodic tasks are scheduled on a fixed-priority basis. Conceptually, it is similar to Chetto and Chetto's algorithm [1]: Sporadic requests are scheduled on the FIFO basis whenever the execution of periodic requests can be postponed. It differs from Chetto and Chetto's algorithm in implementation. Its complexity is linear in the number of periodic tasks while the latter is a pseudo-polynomial time algorithm.

The probability of arrival A_i of requests of each sporadic task can be chosen so that the total maximum utilization of all periodic tasks P_i, for $i = 1, 2, \cdots, N$, is equal to 1. Therefore, the probability of any request in T_i missing its deadline is at most A_i, when the relative deadline of each task is equal to or larger than its period. Because the computation times of periodic requests $P_{i,j}$'s may be less than the maximum value c_i and the slack stealer can make use of the processor time not used by periodic tasks, this probability may actually be smaller than A_i.

To compare the two approaches, namely, RM and sporadic server algorithms and EDF and slack stealing algorithms, to scheduling the transformed tasks, we simulated our example application system for $A = 30\%$ and $A = 20\%$. The average response times and probabilities of missed deadlines of all tasks are listed in Tables B1 and B2 in Appendix B. When EDF and slack stealing algorithms are used, the average response times are better for most tasks and the probabilities of missed deadlines are better for all tasks except task 0. Although the average response time of task 0 is larger when the RM and sporadic server algorithms are used, it does not miss any deadline. In contrast, the probability of its missing deadlines is non-zero when the EDF and slack stealing algorithms are used. We have not yet found any consistent rule to predict what combination of task parameters may lead to poorer performance of the EDF and slack stealing algorithms. However, the overall performance of EDF and slack stealing algorithms is better than the performance of RM and sporadic server algorithms.

5 Summary and Future Work

It has often been said that the well-known periodic-task model does not accurately characterize many real-time applications. One of the reasons is that the maximum computation times of requests in some tasks may be much larger than the corresponding average computation times. This is the case with the semi-periodic tasks examined in this paper. However, despite the fact that the periodic-task model does not accurately characterize a system of semi-periodic tasks, we have shown here that the scheduling algorithms and schedulability analysis methods developed in the framework of the periodic-task model can nevertheless be applied with little modification to schedule semi-periodic tasks and provide them with meaningful performance guarantee.

Specifically, in this paper, we study the problem of providing performance guarantees when the total maximum utilization of the semi-periodic tasks assigned to a processor is larger than one. Hence, the existing schedulability conditions cannot guarantee that the tasks are schedulable, even though their total average utilization is very small. We describe two methods of providing probabilistic schedulability guarantees for the semi-periodic tasks. The probabilistic time-demand analysis method is an extension of the exact schedulability test for periodic tasks. The extension allows us to take into account the variations in the computation times. The transform-task method transforms each semi-periodic task into a periodic task followed by a sporadic task. It allows us to provide an absolute guarantee for requests with shorter computation times and a probabilistic guarantee for longer requests.

Instead of large variations in computation times, the lengths of time between the releases of requests in some tasks may vary widely (but not as widely as those of sporadic or aperiodic tasks). The total maximum utilization of the tasks on one processor, using the minimum time between release times of consecutive requests of each task as the period of the task, may exceed one, and hence the existing schedulability conditions cannot provide any guarantee to the tasks. An analogous method of transforming the tasks into periodic tasks and sporadic tasks can be used in this case. Based on the probability density function of the interarrival times of each task, we can select a threshold value. This value is used as the period of the periodic portion of the task. Most of the requests in this task have interarrival times longer than this threshold value. Requests with interarrival times shorter than this threshold are considered to be sporadic requests. Probabilistic schedulability guarantees similar to those in this paper can then be made to these tasks. It is straightforward to unify this approach with the approach discussed earlier: There is a sporadic request whenever a request $T_{i,j}$ has computation time larger than the computation time c_i of P_i or is released earlier than p_i units of time since the release time of $T_{i,j-1}$.

References

[1] H. Chetto and M. Chetto, "Some Results of the Earliest Deadline Scheduling Algorithm," *IEEE Transactions on Software Engineering*, vol. 15(10), pp. 1261–1269, Oct. 1989.

[2] P. S. Heidmann, "A Statistical Model for Designers of Rate Monotonic Systems," tech. rep., RMA Users Forum, Software Engineering Institute, Pittsburgh PA, 1994.

[3] K. Kant, *Introduction to Computer System Performance Evaluation.* New York: McGraw-Hill, pp. 544, 1992.

[4] J. P. Lehoczky and S. Ramos-Thuel, "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems," in *Proceedings of the IEEE Real-Time System Symposium*, pp. 110–123, 1992.

[5] J. Lehoczky, L. Sha, and Y. Ding, "The Rate Monotonic Scheduling Algorithm – Exact Characterization and Average Case Behavior," in *Proceedings of the IEEE Real-Time System Symposium*, pp. 166–171, 1989.

[6] J. P. Lehoczky, L. Sha, J. K. Strosnider, and H. Tokuda, "Fixed Priority Scheduling Theory for Hard Real-Time Systems," in *Foundations of Real-Time Computing, Scheduling and Resource Management* (A. M. Tilborg and G. M. Koob, eds.), ch. 1, Kluwer Academic Publishers, 1991.

[7] J. Y.-T. Leung and J. Whitehead, "On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks," *Performance Evaluation*, vol. 2, pp. 237–250, 1982.

[8] J. W.-S. Liu, Z. Deng, M. Shankar, M. Storch, J. Sun, T.-S. Tia, and L.-C. Wu, "The Use of PERTS for an Architecture and Timing Study," Tech. Rep. in preparation (available upon request), Department of Computer Science, University of Illinois at Urbana-Champaign, 1995.

[9] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment," in *J. Assoc. Comput. Mach.*, vol. 20(1), pp. 46–61, 1973.

[10] J. W. S. Liu, J. L. R. Redondo, Z. Deng, T. S. Tia, R. Bettati, A. Silberman, M. Storch, R. Ha, and W. K. Shih, "PERTS: A Prototyping Environment for Real-Time Systems," in *Proceedings of the Real-Time System Symposium*, pp. 184–188, 1993.

[11] S. Ramos-Thuel and J. P. Lehoczky, "On-Line Scheduling of Hard Deadline Aperiodic Tasks in Fixed-Priority Systems," in *Proceedings of the IEEE Real-Time System Symposium*, pp. 160–171, 1993.

[12] B. Sprunt, L. Sha, and J. P. Lehoczky, "Aperiodic Task Scheduling for Hard Real-Time Systems," *Real-Time Systems: The International Journal of Time-Critical Computing Systems*, vol. 1, pp. 27–60, 1989.

[13] T.-S. Tia, J. W.-S. Liu, and M. Shankar, "Algorithms and Optimality of Scheduling Soft Aperiodic Requests in Fixed-Priority Preemptive Systems," *To appear in Real-Time Systems: The International Journal of Time-Critical Computing Systems.*

[14] T.-S. Tia, J. W.-S. Liu, J. Sun, and R. Ha, "A Linear-Time Optimal Acceptance Test for Scheduling of Hard Real-Time Tasks," *Submitted to IEEE Transaction on Software Engineering*, 1994.

Appendix A: Response Times of Sporadic Tasks When Scheduled According to the RM and Sporadic Server Algorithms

On Processor 1: Task 18 is periodic with period 6,000,000.

Sporadic Server 1: period 638, exe. budget 200.

Client(s)	period
Task 2	680

Sporadic Server 2: period 2960, exe. budget 442.

Client(s)	period
Task 1	6008
Task 10	3004
Task 27	5260

Sporadic Server 3: period 18212, exe. budget 800.

Client(s)	period
Task 5	18214
Task 20	26019

Sporadic Server 4: period 50064, exe. budget 3000.

Client(s)	period
Task 6	55762
Task 9	50083

On Processor 2:

Sporadic Server 1: period 3047, exe. budget 800.

Client(s)	period
Task 16	3553

Sporadic Server 2: period 30417, exe. budget 16000.

Client(s)	period
Task 3	99667
Task 29	30549
Task 0	3057

On Processor 3:

Sporadic Server 1: period 1738, exe. budget 170.

Client(s)	period
Task 4	2229
Task 12	3715
Task 15	3623
Task 19	3069
Task 22	1738
Task 23	4373
Task 28	3696

Sporadic Server 2: period 7797, exe. budget 500.

Client(s)	period
Task 11	7802
Task 14	8105
Task 17	8190

Sporadic Server 3: period 13446, exe. budget 1340.

Client(s)	period
Task 7	14655
Task 8	14177
Task 21	27932
Task 24	13458

Sporadic Server 4: period 50023, exe. budget 6050.

Client(s)	period
Task 13	96774
Task 25	104529
Task 26	50083

Appendix B: Response Times of Sporadic Tasks When Scheduled According to the Transform Task Method

Table B1: $A = 30\%$

On Processor 1: Task 18 is periodic with period 6,000,000

Task	RM with Sporadic Server		EDF with Slack Stealer	
	ave. response time	% missed deadline	ave. response time	% missed deadline
Task 1	65.69	0	46.09	0
Task 2	157.04	0.95	141.45	0.064
Task 5	18.55	0	10.89	0
Task 6	9.56	0	5.00	0
Task 9	102.53	0	65.41	0
Task 10	199.31	0	150.79	0
Task 20	21.01	0	15.92	0
Task 27	144.58	0	95.87	0

On Processor 2:

Task	RM with Sporadic Server		EDF with Slack Stealer	
	ave. response time	% missed deadline	ave. response time	% missed deadline
Task 0	434.57	0	35.61	0.121
Task 3	43.05	0	29.00	0
Task 16	821.17	2.04	812.23	0.799
Task 29	31.55	0	421.07	0

On Processor 3:

Task	RM with Sporadic Server		EDF with Slack Stealer	
	ave. response time	% missed deadline	ave. response time	% missed deadline
Task 4	96.14	0	79.63	0
Task 7	160.39	0	136.00	0
Task 8	61.06	0	17.29	0
Task 11	49.45	0	38.10	0
Task 12	146.71	0	58.50	0
Task 13	24.57	0	19.00	0
Task 14	31.17	0	46.73	0
Task 15	21.15	0	12.65	0
Task 17	94.42	0	81.17	0
Task 19	29.99	0	31.86	0
Task 21	5.59	0	3.50	0
Task 22	8.14	0	9.03	0
Task 23	22.91	0	18.93	0
Task 24	60.97	0	57.94	0
Task 25	119.83	0	194.00	0
Task 26	248.21	0	256.50	0
Task 28	139.16	0	108.91	0

Table B2: $A = 20\%$

On Processor 1: Task 18 is periodic with period 6,000,000

Task	RM with Sporadic Server		EDF with Slack Stealer	
	ave. response time	% missed deadline	ave. response time	% missed deadline
Task 1	65.24	0	50.11	0
Task 2	167.16	0.76	142.40	0.052
Task 5	16.33	0	12.42	0
Task 6	7.42	0	5.00	0
Task 9	86.48	0	0.00	0
Task 10	142.88	0	107.27	0
Task 20	19.71	0	15.85	0
Task 27	134.95	0	115.56	0

On Processor 2:

Task	RM with Sporadic Server		EDF with Slack Stealer	
	ave. response time	% missed deadline	ave. response time	% missed deadline
Task 0	634.96	0	44.82	0.120
Task 3	49.61	0	29.00	0
Task 16	834.50	1.39	801.95	0.819
Task 29	29.94	0	619.57	0

On Processor 3:

Task	RM with Sporadic Server		EDF with Slack Stealer	
	ave. response time	% missed deadline	ave. response time	% missed deadline
Task 4	77.21	0	69.52	0
Task 7	180.42	0	164.33	0
Task 8	62.63	0	69.07	0
Task 11	46.91	0	39.88	0
Task 12	134.32	0	125.19	0
Task 13	21.89	0	19.00	0
Task 14	38.35	0	37.00	0
Task 15	17.85	0	16.29	0
Task 17	90.28	0	79.06	0
Task 19	44.40	0	44.93	0
Task 21	4.87	0	3.69	0
Task 22	8.95	0	8.38	0
Task 23	24.99	0	19.78	0
Task 24	49.16	0	44.26	0
Task 25	221.33	0	193.00	0
Task 26	249.54	0	170.24	0
Task 28	176.37	0	159.97	0