Report Number: WUCSE-2008-3

2008-01-01

A Practical Schedulability Analysis for Generalized Sporadic Tasks in Distributed Real-Time Systems

Yuanfang Zhang, Donald K. Krecker, Christopher Gill, Chenyang Lu, and Guatam H. Thakar

Existing off-line schedulability analysis for real-time systems can only handle periodic or sporadic tasks with known minimum inter-arrival times. Modeling sporadic tasks with fixed minimum inter-arrival times is a poor approximation for systems in which tasks arrive in bursts, but have longer intervals between the bursts. In such cases, schedulability analysis based on the existing sporadic task model is pessimistic and seriously overestimates the task's time demand. In this paper, we propose a generalized sporadic task model that characterizes arrival times more precisely than the traditional sporadic task model, and we develop a corresponding schedulability analysis that computes tighter... **Read complete abstract on page 2.**

A Practical Schedulability Analysis for Generalized Sporadic Tasks in Distributed Real-Time Systems

Yuanfang Zhang, Donald K. Krecker, Christopher Gill, Chenyang Lu, and Guatam H. Thakar

Complete Abstract:

Existing off-line schedulability analysis for real-time systems can only handle periodic or sporadic tasks with known minimum inter-arrival times. Modeling sporadic tasks with fixed minimum inter-arrival times is a poor approximation for systems in which tasks arrive in bursts, but have longer intervals between the bursts. In such cases, schedulability analysis based on the existing sporadic task model is pessimistic and seriously overestimates the task's time demand. In this paper, we propose a generalized sporadic task model that characterizes arrival times more precisely than the traditional sporadic task model, and we develop a corresponding schedulability analysis that computes tighter bounds on worst-case response times. Experimental results show that when arrival time jitter increases, the new analysis more effectively guarantees schedulability of sporadic tasks.

2008-3

A Practical Schedulability Analysis for Generalized Sporadic Tasks in Distributed Real-Time Systems

Authors: Yuanfang Zhang, Donald K. Krecker, Christopher Gill, Chenyang Lu and Gautam H. Thaker

Corresponding Author: yfzhang@cse.wustl.edu, cdkrecker@atl.lmco.com, cdgill@cse.wustl.edu,

Abstract: Existing off-line schedulability analysis for real-time systems can only handle periodic or sporadic tasks with known minimum inter-arrival times. Modeling sporadic tasks with fixed minimum inter-arrival times is a poor approximation for systems in which tasks arrive in bursts, but have longer intervals between the bursts. In such cases, schedulability analysis based on the existing sporadic task model is pessimistic and seriously overestimates the task's time demand. In this paper, we propose a generalized sporadic task model that characterizes arrival times more precisely than the traditional sporadic task model, and we develop a corresponding schedulability analysis that computes tighter bounds on worst-case response times. Experimental results show that when arrival time jitter increases, the new analysis more effectively guarantees schedulability of sporadic tasks.

Type of Report: Other

A Practical Schedulability Analysis for Generalized Sporadic Tasks in Distributed Real-Time Systems [*]

Yuanfang Zhang[1], Donald K. Krecker[2], Christopher Gill[1], Chenyang Lu[1], and Gautam H. Thaker[2]

[1]Washington University St. Louis, MO, USA {yfzhang, cdgill, lu}@cse.wustl.edu

[2]Lockheed Martin Advanced Technology Laboratories Cherry Hill, NJ, USA {dkrecker, gthaker}@atl.lmco.com

Abstract

Existing off-line schedulability analysis for real-time systems can only handle periodic or sporadic tasks with known minimum inter-arrival times. Modeling sporadic tasks with fixed minimum inter-arrival times is a poor approximation for systems in which tasks arrive in bursts, but have longer intervals between the bursts. In such cases, schedulability analysis based on the existing sporadic task model is pessimistic and seriously overestimates the task's time demand. In this paper, we propose a generalized sporadic task model that characterizes arrival times more precisely than the traditional sporadic task model, and we develop a corresponding schedulability analysis that computes tighter bounds on worst-case response times. Experimental results show that when arrival time jitter increases, the new analysis more effectively guarantees schedulability of sporadic tasks.

1 Introduction

In hard real-time systems, meeting time constraints is crucial, as missing deadlines can cause disastrous failures. As a consequence, providing reliable certification to those systems is essential. Hard real-time applications typically make use of schedulability analysis to guarantee the schedulability of all hard real-time tasks. The analysis can be done off-line before the system executes, and the analysis is based on the knowledge of the release times and the execution times of all tasks. This approach is useful when the system is deterministic, meaning that the release times and the execution times of all tasks are known, and either do not vary or vary only slightly.

Although off-line schedulability analysis is widely used in real-time systems, the existing analysis can only handle periodic tasks and sporadic tasks with known minimum inter-arrival times. The time demand of a sporadic task is treated as that of a periodic task whose period is the minimum inter-arrival time. This can seriously overestimate the task's time demand, especially if it arrives in bursts, and lead to unnecessarily pessimistic upper bounds on its own and other tasks' worst case response times.

We found in practice that instances of a sporadic task usually have a bounded instantaneous arrival rate and a slower average arrival rate. For example, an aircraft tracking application where a group of aircraft appear on the scene nearly simultaneously may generate a burst of tracking jobs. Subsequent track updates for these aircraft also may be in bursts, but only after some longer interval between the bursts. Such sporadic tasks are best defined with at least two constraints. One is the higher instantaneous arrival rate that bounds the maximum number of arrivals over some small time interval. The other is the lower average arrival rate that can also be specified as a maximum number of arrivals over some longer interval, but with a smaller ratio of arrivals per unit time. In this paper, we call sporadic tasks defined with multiple constraints *generalized sporadic tasks*, and we refer to sporadic tasks with only a minimum inter-arrival time constraint *traditional sporadic tasks*.

Research Contributions: In this work, we have (1) defined a generalized sporadic task model, which improves on the traditional sporadic task model by characterizing arrival times more precisely; (2) developed a new off-line schedulability analysis that computes tighter bounds on worst-case response times for applications with generalized sporadic tasks; (3) extended the release guard synchronization protocol to govern the release of end-to-end generalized sporadic tasks as well as end-to-end periodic tasks; (4) designed separate end-to-end schedulability analysis algorithms for the direct synchronization protocol and for the generalized release guard synchronization protocol; and (5) conducted experiments based on realistic workloads. The results of these experiments show that our generalized sporadic schedulability analyses significantly tighten the bounds on worst-case response times and more effectively guarantee schedulability of sporadic tasks when arrival-time jitter increases.

[*]Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

2 Generalized Sporadic Task Model

In this paper, we treat a sporadic task as a stream of sporadic jobs. To compute the worst-case response time of sporadic jobs in a system with a fixed-priority preemptive scheduling policy, we need to model the constraints on the arrival pattern of the sporadic jobs. The traditional sporadic task model imposes the constraint that some minimum time must elapse between any two arrivals of the task. It admits a worst-case response time analysis by treating the minimum inter-arrival time as the task's period.

Wang et al. [15] adopted the (σ, ρ) leaky bucket filter model used in communication networks to model sporadic tasks. ρ is the token input rate and σ is the bucket size. The filter can hold at most σ tokens at any time and is filled at a constant rate of ρ tokens per unit time. A sporadic task that follows the (σ, ρ) leaky bucket model will generate its jobs as follows. The filter releases a job J_k with execution time C_k when it has at least C_k tokens. C_k tokens are removed from the filter after the job J_k is released. No job can be released when the filter does not have enough tokens for the job execution time. As a result, the bucket size σ should be at least the maximum execution time among all jobs of the task so that they may enter the system. From this definition, we can see that a periodic task with a period equal to or larger than σ/ρ and execution time equal to or less than σ satisfies the (σ, ρ) leaky bucket model. In general, each task can be modeled by many (σ, ρ) pairs once the workload from the task in any time interval $[t_1, t_2]$ is never larger than $\sigma + \rho * (t_2 - t_1)$.

Although the leaky bucket model may model the workload entering the system from any task, it is more suitable for modeling periodic tasks with fixed inter-arrival times. For sporadic tasks that may arrive in bursts, the leaky bucket model can not provide precise upper bounds on their workloads. This imprecision impairs schedulability analysis of the system. To model the arrival pattern of sporadic jobs more precisely, we present a new practical model for sporadic tasks when scheduling them on a standard real-time operating system.

The traditional sporadic task model can be interpreted in a different way. The minimum inter-arrival time constraint can be understood as a sliding time window of the same fixed length, each instance of which can have at most one arrival of the task. This constraint can be generalized in two ways: (1) by introducing a limit greater than one on the number of arrivals allowed in a time window or (2) by allowing multiple pairs of windows and limits on the number of arrivals. Thus a generalized sporadic task T_i may be characterized by a set of K(i) arrival time constraints

$\{(z_{i,k}, w_{i,k}) 1 \le k \le K(i)\}$

where at most $z_{i,k}$ arrivals of T_i occur in any window of length $w_{i,k}$, both $z_{i,k}$ and $w_{i,k}$ are strictly increasing as k increases, and $z_{i,k}$ is a natural number.

For example, a generalized sporadic task T_i with three arrival constraints $K(i) = 3$ $\{(z_{i,1} = 1, w_{i,1} = 2), (z_{i,2} = 3, w_{i,2} = 10), (z_{i,3} = 5, w_{i,3} = 18)\}$ could have arrivals at times 0, 2, 4, 10, 12, 18, 20, 22, 28, 30, 36, 38, 40, 46, 48, 54, 56, 58, 64, ...

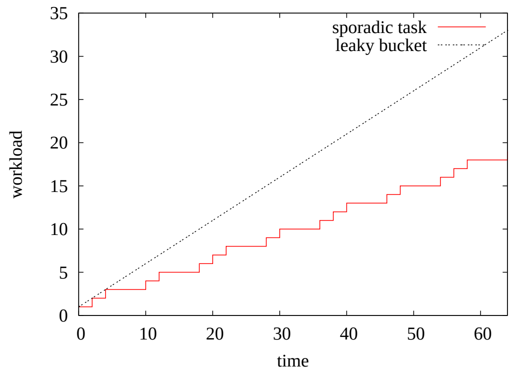

Figure 1. Leaky bucket vs. generalized sporadic task

Assuming the execution time of T_i is one unit, we plotted in Figure 1 the workload entering the system from T_i modeled by a leaky bucket model with $\sigma = 1$, $\rho = 0.5$ and treated as a generalized sporadic task with all 3 of its constraints. There is a big gap between the leaky bucket workload and the generalized sporadic task workload. The greatly overestimated workload can adversely affect analysis of the schedulability of T_i and any other tasks in the same system.

3 Schedulability Analysis for Generalized Sporadic Tasks

3.1 Time-Demand Analysis for Generalized Sporadic Tasks

The definition of a **level-i busy period** [6] is: a time interval $(t_0, t_0 + t]$ within which jobs with priority i or higher are processed through $(t_0, t_0 + t]$, but no jobs of level i or higher are processed in $(t_0 - \varepsilon, t_0]$ or $(t_0 + t, t_0 + t + \varepsilon)$ for sufficiently small $\varepsilon > 0$.

Suppose T_i is a generalized sporadic task of priority i. All response times of the jobs of task T_i are a part of some level-i busy period. The longest response time for a job occurs during a level-i busy period $(t_0, t_0 + t]$ if the arrivals of all tasks with equal or higher priority satisfy the maximum number of arrivals in that level-i busy interval.

We want to determine the maximum number of arrivals that a task can generate over any interval $(t_0, t_0 + t]$. If the task is periodic or traditional sporadic, the number is greatest when one instance of the task is released at t_0 and subsequent instances are released after every integer multiple of the period (or minimum inter-arrival time). However, if the task is generalized sporadic, the maximum number of arrivals accumulates when each new release occurs as early as the set of constraints $\{(z_{i,k}, w_{i,k}) 1 \le k \le K(i)\}$ allows. The *maximum number of arrivals function* of the generalized sporadic task T_i, $MNA_i(t)$, for any interval $(t_0, t_0 + t]$ can be expressed by the following recursive definition:

$$MNA_i(t) = \begin{cases} 0 & \text{if } t \le 0 \\ min\{MNA_i(t - w_{i,k}) + z_{i,k}\} & \text{if } t > 0 \end{cases},$$

where $1 \le k \le K(i)$.

Proof: For any time interval t, then for every k, $1 \le k \le K(i)$, the time interval can be divided into two separate parts; the length of the first part is $w_{i,k}$, and the length of the second part is $t - w_{i,k}$. The maximum number of arrivals in interval t should be no more than the sum of the maximum numbers of arrivals in those two parts, since neither part can accept any extra arrivals. So $MNA_i(t) \le MNA_i(w_{i,k}) + MNA_i(t - w_{i,k}) = z_{i,k} + MNA_i(t - w_{i,k})$.

If $t < w_{i,k}$, the length of the first part is t, and the length of the second part is 0. Then $MNA_i(t) \le z_{i,k}$, so the above inequality still holds. Taking the minimum of the right hand side for all k,

$MNA_i(t) \le min\{MNA_i(t - w_{i,k}) + z_{i,k} | 1 \le k \le K(i)\}$

Let τ_i be the maximum execution time of all instances of task T_i. The *maximum time demand function* of the generalized sporadic task T_i, $TD_i(t)$, for any interval $(t_0, t_0 + t]$, is computed from the maximum number of arrivals function $MNA_i(t)$ and is given by

$TD_i(t) = MNA_i(t) * \tau_i$.

The *earliest arrival time function* of the n^{th} job of the generalized sporadic task T_i, $EAT_i(n)$, for time t in any interval $(t_0, t_0 + t]$ can be expressed by the following recursive definition:

$$EAT_i(n) = \begin{cases} -\infty & \text{if } n \le 0 \\ 0 \quad \text{if } n > 0 \ \ and & n \le z_{i,1} \\ max\{EAT_i(n - z_{i,k}) + w_{i,k}\} & \text{if } n > z_{i,1} \end{cases},$$

where $1 \le k \le K(i)$.

3.2 Schedulability Test for Generalized Sporadic Tasks on a Single Processor

For tasks on a single processor, our schedulability analysis tests one task at a time starting from the highest priority task T_1 in decreasing order of priority. For the purpose of determining whether a task T_i is schedulable, we assume, without loss of generality, that the level-i busy period begins at time 0. Let H_i be the set of higher or equal priority tasks assigned to the same processor as task T_i but excluding task T_i. The following steps establish whether or not T_i is schedulable:

(1) Compute an upper bound D_i on the duration of a level-i busy period

$$D_i = min\{t > 0 | t = \sum_{T_j \in H_i \cup T_i} MNA_j(t) * \tau_j\} \qquad (1)$$

(2) Compute an upper bound M_i on the number of instances of T_i in a level-i busy period of duration D_i

$M_i = MNA_i(D_i)$.

(3) For $m = 1$ to M_i, do

(a) Compute an upper bound $C_i(m)$ on the completion time of m^{th} job of T_i

$C_i(m) = min\{t > 0 | t = \sum_{T_j \in H_i} MNA_j(t) * \tau_j + m * \tau_i\}$.

(b) Since a lower bound for the release time of the m^{th} job of T_i is $EAT_i(m)$, compute an upper bound to the response time of the m^{th} job of T_i in the busy period

$V_i(m) = C_i(m) - EAT_i(m)$.

(4) Compute the WCRT for T_i by

$W_i = max\{V_i(m)\}, \ for \ 1 \le m \le M_i$.

(5) If W_i is not greater than the task's deadline, T_i is schedulable. Otherwise, it is unschedulable.

Generalized Sporadic Task Utilization Test: Equation 1 may not have a finite solution if the processor is overloaded with higher or equal priority tasks. Theorem 1 provides a sufficient utilization test that the processor is not overloaded. The proof of this theorem makes use of the following lemma.

Lemma 1: Suppose that for the generalized sporadic task T_i, the constraint with the smallest limit/length ratio is $z_{i,kmin}/w_{i,kmin} = min\{z_{i,j}/w_{i,j} | 1 \le j \le K(i)\}$. Then $\frac{z_{i,kmin}}{w_{i,kmin}} * t \le MNA_i(t) \le \frac{z_{i,kmin}}{w_{i,kmin}} * t + z_{i,kmin}$

Proof:

Induction basis: When $0 < t$, by the constraint $z_{i,j} \ge 1$ for any j, $1 \le j \le K(i)$, thus at least one arrival can occur in the interval $(0, t)$. When $t \le w_{i,kmin}/z_{i,kmin}$, since $w_{i,kmin}/z_{i,kmin} \le w_{i,kmin}$, by the definition of the generalized sporadic constraints, at most $z_{i,kmin}$ arrivals can occur in the interval $(0, w_{i,kmin}/z_{i,kmin})$. Combining the above two conditions, when $0 < t \le w_{i,kmin}/z_{i,kmin}$, $1 \le MNA_i(t) \le z_{i,kmin}$. So the Lemma holds, when $0 < t \le w_{i,kmin}/z_{i,kmin}$.

Induction hypothesis: Suppose when $t \le T$, the lemma holds.

Induction Step: When $T < t < T + w_{i,1}$, we first prove the lower bound is satisfied.

For every j, $1 \le j \le K(i)$, substitution of $t - w_{i,j}$ for t and transposing gives

$$\frac{z_{i,kmin}}{w_{i,kmin}} t \leq MNA_i(t - w_{i,j}) + \frac{z_{i,kmin}}{w_{i,kmin}} w_{i,j}$$

Since $\frac{z_{i,kmin}}{w_{i,kmin}} \leq \frac{z_{i,j}}{w_{i,j}}$,

$$\frac{z_{i,kmin}}{w_{i,kmin}} t \leq MNA_i(t - w_{i,j}) + z_{i,j}$$

Taking the minimum of the right-hand side for all j,

$$\frac{z_{i,kmin}}{w_{i,kmin}} t \leq MNA_i(t)$$

Then we prove the upper bound is also satisfied, when $T < t < T + w_{i,1}$. Substitution of $t - w_{i,kmin}$ for t and simplifying gives

$$MNA_i(t - w_{i,kmin}) \leq \frac{z_{i,kmin}}{w_{i,kmin}} t$$

By the definition of the generalized sporadic constraints, at most $z_{i,kmin}$ arrivals can occur in the interval $[t - w_{i,kmin}, t)$. Therefore,

$$\begin{aligned} MNA_i(t) &\leq MNA_i(t - w_{i,kmin}) + z_{i,kmin} \\ &\leq \frac{z_{i,kmin}}{w_{i,kmin}} t + z_{i,kmin} \end{aligned}$$

A periodic task T_i with period p_i is a special case of a generalized sporadic task with a single arrival time constraint $(1, p_i)$. By Lemma 1, this also satisfies

$$\frac{1}{p_i} t \leq MNA_i(t) \leq \frac{1}{p_i} t + 1$$

Let a set of fixed-priority generalized sporadic tasks $\{T_j\}$ be assigned to a processor. With respect to a specific task T_i, let PT_i be the set of periodic tasks in $H_i \cup T_i$, and let ST_i be the set of (non- periodic) generalized sporadic tasks in $H_i \cup T_i$. For each T_j in ST_i, let $z_{j,k_j}/w_{j,k_j}$ be its smallest limit/length constraint ratio (the minimizing ratio depends on j). The level-i generalized sporadic task utilization is defined as

$$\sum_{T_j \in PT_i} \frac{\tau_j}{p_j} + \sum_{T_j \in ST_i} \frac{\tau_j * z_{j,k_j}}{w_{j,k_j}}$$

Theorem 1: If a processor is assigned a set of fixed-priority generalized sporadic tasks, and if the level-i generalized sporadic task utilization is less than 1, then any level-i busy period on the processor has finite duration.

Proof: By taking Lemma 1 for each task T_j, multiplying the inequality by τ_j, and summing,

$$\begin{aligned} (\sum_{T_j \in PT} \frac{\tau_j}{P_j} + \sum_{T_j \in ST} \frac{\tau_j * z_{j,k}}{w_{j,k}}) * t &\leq \sum_{T_j \in H_i \cup T_i} MNA_j(t) * \tau_j \\ &\leq (\sum_{T_j \in PT} \frac{\tau_j}{P_j} + \sum_{T_j \in ST} \frac{\tau_j * z_{j,k}}{w_{j,k}}) * t \\ &+ \sum_{T_j \in PT} \tau_j + \sum_{T_j \in ST} z_{j,k} * \tau_j \end{aligned}$$

Since the level-i generalized sporadic task utilization < 1, the right side of this inequality grows less rapidly than t and is less than t for sufficiently large t, $(\sum_{T_j \in PT} \frac{\tau_j}{P_j} + \sum_{T_j \in ST} \frac{\tau_j * z_{j,k}}{w_{j,k}}) * t + \sum_{T_j \in PT} \tau_j + \sum_{T_j \in ST} z_{j,k} < t$ and

For sufficiently small t, $\sum_{T_j \in H_i \cup T_i} MNA_j(t) * \tau_j = \sum_{T_j \in PT} \tau_j + \sum_{T_j \in ST} z_{j,k} > t$

So $t = \sum_{T_j \in H_i \cup T_i} MNA_j(t) * \tau_j$ will have a finite solution.

3.3 Schedulability Test for End-to-End Generalized Sporadic Tasks via Generalized Release Guards

Given a set of generalized sporadic end-to-end tasks, each task is comprised of a linear chain of subtasks $T_{i,1}, T_{i,2}, ..., T_{i,n(i)}$ where each subtask $T_{i,j}$ belonging to task T_i may be assigned to a different processor P_j.

Although each generalized sporadic task T_i has arrival time constraints, these constraints apply a priori only to each initial subtask $T_{i,1}$ of an end-to-end task, and the inter-release times of consecutive jobs in later subtasks may not satisfy the original time constraints. The clumping effect in the later subtasks caused by the Direct Synchronization protocol (DS) can have an undesirable effect on the schedulability of end-to-end tasks in a priority-driven system [14]. Moreover, the upper bounds on end-to-end response times produced by current algorithms for DS are not tight. However, if we can govern the releases of the jobs in later subtasks and make them follow the task's time constraints, the previously described analysis can be carried out on each processor to determine a worst-case response time $W_{i,j}$ for each subtask $T_{i,j}$. This improves the schedulability of end-to-end tasks greatly.

The generalized sporadic arrival time constraints can be applied to non-initial subtasks by maintaining a **generalized release guard** $g_{i,j}$ for each non-initial subtask $T_{i,j}, (j > 1)$. Let $r_{i,j}(m)$ be the release time and let $C_{i,j}(m)$ be the completion time of the m^{th} job of $T_{i,j}$. The following rules are used to update $g_{i,j}(j > 1)$.

1. At the initial time, set $g_{i,j}$=0.

2. When $m - 1^{th}$ job of $T_{i,j}$ is released at time $r_{i,j}(m-1)$, update $g_{i,j} = r_{i,j}(m-1) + (r_{i,1}(m) - r_{i,1}(m-1))$.

3. Update $g_{i,j}$ to the current time if the current time is a processor idle point on the processor where $T_{i,j}$ executes.

The scheduler releases the m^{th} job of $T_{i,j}$ either at $g_{i,j}$, or at $C_{i,j-1}(m)$ when the immediate predecessor $T_{i,j-1}$ completes its m^{th} job, whichever is later.

The generalized release guard protocol propagates the generalized sporadic arrival time constraints obeyed by the initial subtasks $T_{i,1}$ along the subtask chains so that they are also obeyed by the non-initial subtasks $T_{i,j}$ for $j > 1$. Worst-case response times of all subtasks can be determined as described above, and they can simply be summed to obtain worst-case response times of the end-to-end tasks. This is established by the following theorem and lemma.

Theorem 2: An upper bound W_i to the end-to-end response time of any generalized sporadic task T_i in a fixed-priority system synchronized according to the generalized release guard protocol is given by $W_i = \sum_{k=1}^{n(i)} W_{i,k}$

Here $n(i)$ is the number of subtasks in T_i. $W_{i,k}$ is the upper bound on the response time of the subtask $T_{i,k}$ obtained by considering only subtasks on the same processor as $T_{i,k}$ and treating every such subtask as a generalized sporadic task satisfying its own generalized sporadic arrival time constraints. The theorem is a direct consequence of the following lemma.

Lemma 2: When subtasks are synchronized according to the generalized release guard protocol, every job of every subtask $T_{i,k}$ for k=2,3,...,n(i) is released no later than $\sum_{l=1}^{k-1} W_{i,l}$ units of time after the release time of the corresponding job in its first sibling subtask $T_{i,1}$.

Proof: According to the definition of the generalized release guard protocol, for every $k = 2, 3, ..., n(i)$, the first job of subtask $T_{i,k}$ is released when the first job of its immediate predecessor $T_{i,k-1}$ completes, and this is surely within $W_{i,k-1}$ units of time after the release of $T_{i,k-1}$. Hence, for any $2 \leq k \leq n(i)$, the first job in $T_{i,k}$ is released by $\sum_{l=1}^{k-1} W_{i,l}$ units of time after the release of the first job in $T_{i,1}$, that is, the lemma is true for the first jobs of all subtasks of T_i.

The lemma is also true for all the jobs in the second sibling subtask $T_{i,2}$. To prove this statement, let us suppose that the lemma is true for all the jobs of $T_{i,2}$ up to and including the x^{th} job, for some $x \geq 1$. Then for the x^{th} job, $r_{i,2}(x) - r_{i,1}(x) \leq W_{i,1}$, $r_{i,2}(x) + r_{i,1}(x+1) - r_{i,1}(x) \leq r_{i,1}(x+1) + W_{i,1}$. Moreover, $C_{i,1}(x+1) \leq r_{i,1}(x+1) + W_{i,1}$. So $r_{i,2}(x+1) = max(C_{i,1}(x+1), r_{i,2}(x) + (r_{i,1}(x+1) - r_{i,1}(x))) \leq r_{i,1}(x+1) + W_{i,1}$ satisfies the lemma.

Now suppose that the lemma is true for the all the jobs in all the predecessor sibling subtasks of $T_{i,k}$ for some k in the range $2 < k \leq n(i)$ and it is also true for the x^{th} job and all the jobs before the x^{th} of $T_{i,k}$. To show the lemma is true for the $(x+1)^{th}$ job of $T_{i,k}$ as well, we first examine the release time of the x^{th} job of $T_{i,k}$. It satisfies $r_{i,k}(x) - r_{i,1}(x) \leq \sum_{l=1}^{k-1} W_{i,l}$. Adding $r_{i,1}(x+1)$ at both sides, $r_{i,k}(x) + r_{i,1}(x+1) - r_{i,1}(x) \leq r_{i,1}(x+1) + \sum_{l=1}^{k-1} W_{i,l}$. Moreover, $C_{i,k-1}(x+1) \leq r_{i,1}(x+1) + \sum_{l=1}^{k-1} W_{i,l}$. So $r_{i,k}(x+1) = max(C_{i,k-1}(x+1), r_{i,k}(x) + (r_{i,1}(x+1) - r_{i,1}(x))) \leq r_{i,1}(x+1) + \sum_{l=1}^{k-1} W_{i,l}$ satisfies the lemma.

3.4 Schedulability Test for End-to-End Generalized Sporadic Tasks via Direct Synchronization Protocol

Although (as our schedulability analysis experiments in Section 4 show), the release guard synchronization protocol (RG) improves schedulability when compared to the direct synchronization protocol (DS), RG is rarely implemented or used in current systems. Therefore we develop a schedulability analysis algorithm that handles DS for generalized sporadic tasks.

Our approach follows that of Sun's SA/DS (Schedulability Analysis for Direct Synchronization Protocol) algorithm [14] for periodic tasks. We first provide an algorithm that is used iteratively to calculate an upper bound on the intermediate end-to-end response time (IEERT) of each subtask $T_{i,j}$. The IEERT of $T_{i,j}$ is the maximum time between the release of a job in $T_{i,1}$ and the completion of the corresponding job in $T_{i,j}$. The end-to-end response time of the entire task T_i is the IEERT of its last subtask $T_{i,n(i)}$. An upper bound on the IEERT of the last subtask is thus an upper bound on the end-to-end response time of the task.

We follow a variation of Sun's IEERT algorithm that uses minimum execution times $\sigma_{i,j}$ along with maximum execution times $\tau_{i,j}$ to give tighter upper bounds on IEERTs. Let $S_{i,0} = 0$ and $S_{i,j} = sum\{\sigma_{i,k} | 1 \leq k \leq j\}$ for $j > 0$. If the current upper bound on intermediate end-to-end response time of $T_{i,j-1}$ is $V_{i,j-1}$, an upper bound on the "jitter" in the release time of its successor $T_{i,j}$ is $V_{i,j-1} - S_{i,j-1}$. Although the generalized sporadic arrival time constraints do not apply to every subtask in DS, they do apply to the initial subtasks. Thus the values $MNA_{i,1}(t)$ and $EAT_{i,1}(m)$ can be calculated. $H_{i,j}$ is the set of higher or equal priority subtasks assigned to the same processor as $T_{i,j}$, but excluding $T_{i,j}$.

Generalized Sporadic IEERT Algorithm $\{V' = IEERT(T, V)\}$

Input:

1. A set $\{T_i\}$ of end-to-end generalized sporadic tasks
2. A set $\{V_{i,j}\}$ of bounds on the IEERT of subtasks

Output: A set $\{V'_{i,j}\}$ of new bounds on the IEERT of subtasks

Algorithm:

For each generalized sporadic subtask $T_{i,j}$

1. Compute an upper bound $D_{i,j}$ on the duration of a level-(i,j) busy period:

$$D_{i,j} = min\{t > 0 | t = \sum_{T_{u,v} \in H_{i,j} \cup T_{i,j}} MNA_{u,1}(t + V_{u,v-1} - S_{u,v-1}) * \tau_{u,v}\}$$

2. Compute an upper bound $M_{i,j}$ on the number of instances of $T_{i,j}$ in a level-(i,j) busy period of duration $D_{i,j}$: $M_{i,j} = MNA_{i,j}(D_{i,j}) = MNA_{i,1}(D_{i,j} + V_{i,j-1} - S_{i,j-1})$

3. For $m = 1$ to $M_{i,j}$ do

(a) Compute an upper bound on the completion time $C_{i,j}(m)$ of the m^{th} job of $T_{i,j}$

$$C_{i,j}(m) = min\{t > 0 | t = m * \tau_{i,j} + \sum_{T_{u,v} \in H_{i,j}} MNA_{u,1}(t + V_{u,v-1} - S_{u,v-1}) * \tau_{u,v}\}$$

(b)Since a lower bound for the release time of the m^{th} job of $T_{i,1}$ is $EAT_{i,1}(m) - V_{i,j-1}$, compute an upper bound on the IEERT of the m^{th} job of $T_{i,j}$ in the busy period

$V_{i,j}(m) = C_{i,j}(m) + V_{i,j-1} - EAT_{i,1}(m)$

4. Compute the new bound $V^{'}_{i,j}$ by

$V^{'}_{i,j} = max\{V_{i,j}(m)\}\ for\ 1 \le m \le M_{i,j}$

Our algorithm to calculate the upper bounds of end-to-end response times iteratively uses our IEERT algorithm. Its input is a set $\{V^0_{i,j}\}$ of initial estimates of IEERTs of all subtasks in the system. The initial estimated IEERT for the subtask $T_{i,j}$ is the sum of the maximum execution times of $T_{i,j}$ and all its predecessor subtasks. During each iteration, say the $(x+1)^{th}$, the IEERT algorithm uses the set $\{V^x_{i,j}\}$ of estimates produced in the x^{th} iteration as input and produces as output a new set of estimates $\{V^{x+1}_{i,j}\}$. If the output estimate for every subtask is equal to the input estimate for the subtask, then the bound $\{V^{x+1}_{i,j}\}$ produced during the iteration is a correct upper bound on the IEERT of $T_{i,j}$, and an upper bound on the end-to-end response time of the task T_i is equal to $V^{x+1}_{i,n(i)}$. If the input and output estimates for some subtasks are not equal, another iteration is carried out using as input the output estimates just produced.

To prove that when this algorithm terminates, the outputs produced during the last iteration are the correct upper bounds of the end-to-end response times of all tasks, we only need to prove the same theorem as that in Sun's thesis [14]. The proof of this theorem makes use of the same lemma.

Lemma 3: Suppose that a subtask instance $T_{i,j}(m)$ completes at time t. If the IEERT of every subtask instance $T_{u,v}(w)$ that completes before t is no greater than some $V_{u,v} > 0$, then the IEERT of $T_{i,j}(m)$ is no greater than $V^{'}_{i,j}$, where $V^{'} = IEERT(T, V)$.

Proof: The correctness of this Lemma follows from the calculation of the busy period in our IEERT algorithm. Obviously, the execution of any instance $T_{i,j}(m)$ of $T_{i,j}$ must be contained in a level-(i,j) busy period. Without loss of generality, we assume that the level-(i,j) busy period within which $T_{i,j}(m)$ executes starts from time zero, and $T_{i,j}(m)$ is the m_{th} released instance of $T_{i,j}$ in this busy period. According to the condition of Lemma 3, we know an upper bound on the time demand that can be generated by $T_{i,j}$ and other subtask instances that can delay the completion of $T_{i,j}(m)$. In other words, we know a time demand function $TD^{'}(x)$, which is such that $TD^{'}(x) \ge TD(x)$ for $0 < x < t$. The IEERT algorithm uses $TD^{'}(x)$ in its first step $t^{'} = min\{x > 0 | x = TD^{'}(x)\}$. We can verify that $t^{'} \ge t$, i.e., $t^{'}$ is an upper bound on the completion time of $T_{i,j}(m)$. Since we also know a lower bound on the release time of $T_{i,j}(m)$ in the IEERT algorithm, the computed bound on IEERT $V^{'}_{i,j}$, which is the maximum bound on IEERT for all instances of $T_{i,j}$ in that busy period, is also an upper bound on the IEERT of $T_{i,j}(m)$.

Theorem 3: Let $V = V_{i,j}$ be a set of positive numbers, where there is a one-to-one mapping between $V_{i,j}$ and $T_{i,j}$. If $V = IEERT(T, V)$, then $V_{i,j}$ is a correct upper bound on the IEERT time of $T_{i,j}$.

Proof: The proof follows the induction proof [14] from Sun's thesis.

4 Experimental Evaluation

4.1 Comparison of Different Analyses

In this experiment, we assume four end-to-end periodic tasks executing on three processors. Their periods, deadlines, maximum execution times and priorities are shown in Table 1. In addition, minimum execution times of subtasks are equal to the maximum execution times in the table. First, we change task T_3 from being a periodic task to be-

$T_{i,j}$	P_i	**period**	**deadline**	**exec.**	**prio.**
$T_{1,1}$	P_1	312	284	21	4
$T_{1,3}$	P_1	312	284	75	4
$T_{3,1}$	P_1	162	162	30	2
$T_{3,3}$	P_1	162	162	42	2
$T_{2,1}$	P_2	90	90	23	1
$T_{2,3}$	P_2	90	90	30	1
$T_{4,1}$	P_2	203	203	58	3
$T_{1,2}$	P_3	312	284	24	4
$T_{2,2}$	P_3	90	90	13	1
$T_{3,2}$	P_3	162	162	18	2
$T_{4,2}$	P_3	203	203	20	3

Table 1. Task Settings

ing a generalized sporadic task. To make it easier to understand how WCRTs change, we maintain the same subtask priorities when we convert periodic tasks to sporadic tasks. In the conversion, we make the sporadic behavior a jittery variation of the periodic behavior by allowing arrivals a bit closer together than the period but keeping the same bound on the number of arrivals over a multi-period window. For instance, the time constraints for the converted task T_3 are $\{(1, 113), (2, 324)\}$. These time constraints allow the sporadic task T_3 to arrive once in time window $0.7 * period$, but still only twice in time window $2 * period$. T_3 is allowed 30 percent jitter in this case. T_3 will be made increasingly jittery in some later cases. We are interpreting J percent jitter to mean that T_3 has a first constraint set to $(1, (1 - J/100) * period)$.

We distinguish original and new WCRT analyses. Since the original analysis treats a sporadic task's minimum interarrival time like a period, we call it Periodic-Task analysis (PT). Our new analysis handles generalized sporadic tasks with multiple time constraints and does not interpret them as periodic tasks. Therefore we call it Sporadic-Task analysis (ST). Each of these two analyses has different algorithms for the DS and the RG synchronization protocols. Thus we compare the following four distinct analysis algorithms:
PT/DS: original periodic-task analysis with DS
PT/RG: original periodic-task analysis with RG
ST/DS: new generalized sporadic-task analysis with DS
ST/RG: new generalized sporadic-task analysis with RG

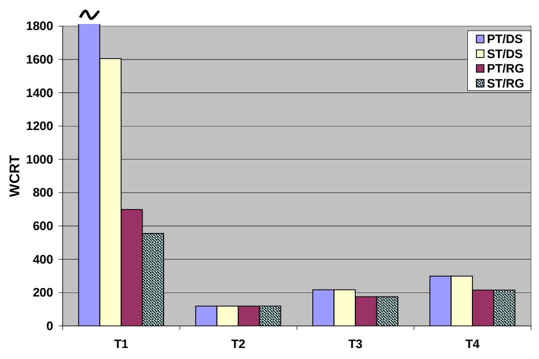

Figure 2. WCRT comparisons for 4 tasks when T_3 is allowed 30 percent jitter

We then converted task T_3 into a generalized sporadic task with time constraints $\{(1, 113), (2, 324)\}$. Figure 2 shows that the WCRTs with RG are no worse than the WCRTs with DS, and they are noticeably better for tasks T_3, T_4, and especially T_1. The WCRT for T_1 in PT/DS is not shown entirely in Figure 2, because it is larger than 40000, which exceeds the upper bound of our analysis. We also can see that the ST analysis outperforms PT on both DS and RG protocols. This is because the PT counts on many more arrivals than the generalized sporadic tasks are

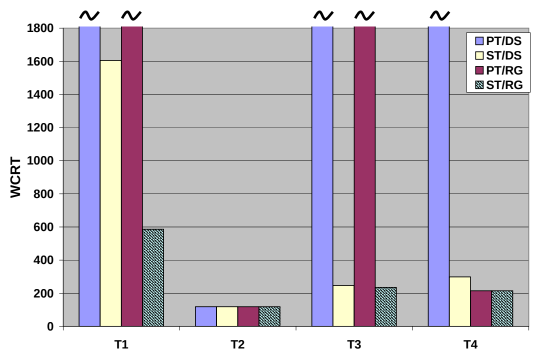

Figure 3. WCRT comparisons for 4 tasks when T_3 is allowed 60 percent jitter

actually allowed. To exhibit the benefit of our ST analysis more clearly, we increased the arrival time jitter of T_3 by changing its time constraints. Figure 3 shows the result when the time constrains for T_3 are $\{(1, 65), (2, 324)\}$. The PT performed even worse. The WCRTs for T_1, T_3, and T_4 in PT/DS are infinite according to utilization bound in [14], as well as the WCRTs for T_1 and T_3 in PT/RG.

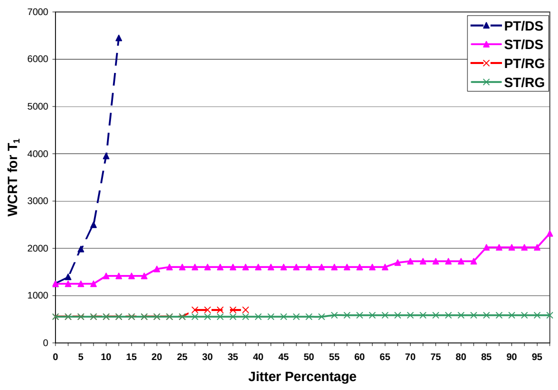

Figure 4. T_1's WCRT bounds as T_3 jitter increases

To further show the relationship between arrival time jitter and the WCRTs calculated by the different analyses, we set the time constraints for T_3 as $\{(1, (1 - x/100) * period), (2, 324)\}$. The horizontal axis in Figure 4 represents jitter increasing from 0 percent to 97.5 percent, as the window size of the first time constraint decreases from the full period to 2.5 percent of it. Then we used four different algorithms to calculate the WCRTs for task T_1, which ran at the lowest priority and was affected greatly by the arrival pattern of T_3. In Figure 4, even to tolerate 12.5 percent jitter, the WCRT calculated by PT/DS is 6450, which is four times more than the WCRT calculated by ST/DS. When the jitter percentage further increases to 15, the WCRT by PT/DS exceeds the upper bound of our analysis. For PT/RG, the calculated WCRTs are much less than PT/DS. However, when

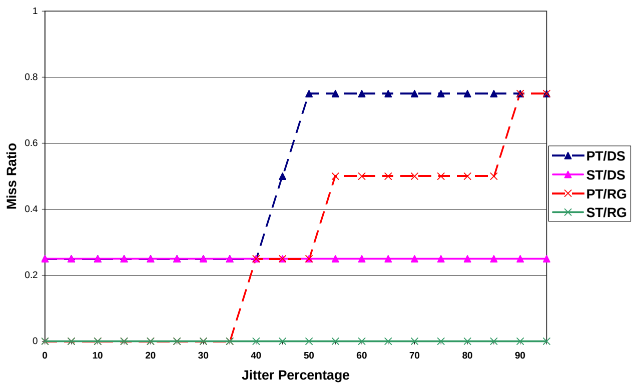

Figure 5. Miss ratio comparisons for four tasks when T_3 is allowed x percent jitter. The time constraints for T_3 are $\{(1,(1-x/100)*162),(2,324)\}$

the jitter percentage reaches 37.5, the WCRT for T_1 is infinite when PT/RG is used. For ST/RG, the WCRTs for T_1 are fairly stable under 600 as the jitter percentage increases from 0 to 97.5. This is because our ST analysis considers all the arrival time constraints. Although the first time constraint becomes tighter and tighter, the arrival pattern of T_3 still needs to satisfy the second time constraint which does not change. The second constraint bounds the arrival numbers of task T_3 and reduces its impact on task T_1.

Figure 5 depicts the miss ratios of the 4 tasks, i.e., what fraction of them can miss their deadlines, as the arrival time jitter of task T_3 increases from 0 to 95 percent along the horizontal axis. We calculated the WCRT bounds using the 4 different algorithms and compared them with the tasks' deadlines. If the WCRT bound exceeds the deadline, the task is not proved schedulable and is counted in the miss ratio. The miss ratios that are calculated by the ST analysis are fairly stable while the miss ratios that are calculated by the PT analysis reach 75%.

4.2 Representative Example

Military shipboard computing is moving toward a common computing and networking infrastructure that hosts the mission execution, mission support and quality of life systems required for shipboard operations. One example of a shipboard computing system model includes 15 end-to-end periodic tasks. Each task consists of varying number of subtasks between 5 and 15. Altogether there are 150 subtasks allocated across 50 processors. The aggregate loading of all the 50 processors is about 50%. We convert task T_{10} to be a generalized sporadic task. The original task T_{10} has period 200 and, assuming rate monotonic priorities, has the highest priority along with task T_2. Moreover, task T_{10}'s 12 subtasks share processors with all of the other tasks except for T_4 and T_6. The time constraints for the converted task T_{10} are $\{(1,(1-x/100)*200),(2,400)\}$.

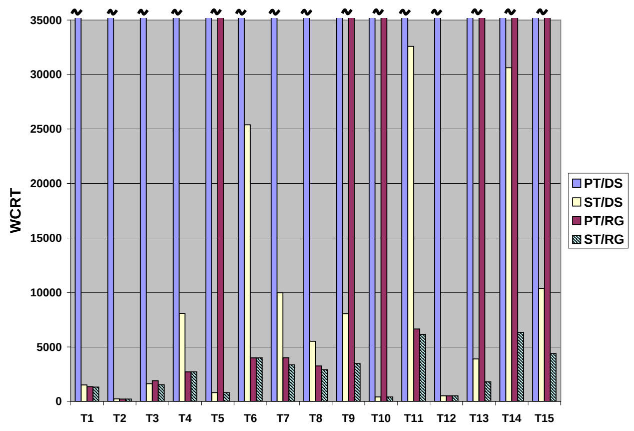

Figure 6. WCRT comparisons for 15 tasks when T_{10} is allowed 75 percent jitter

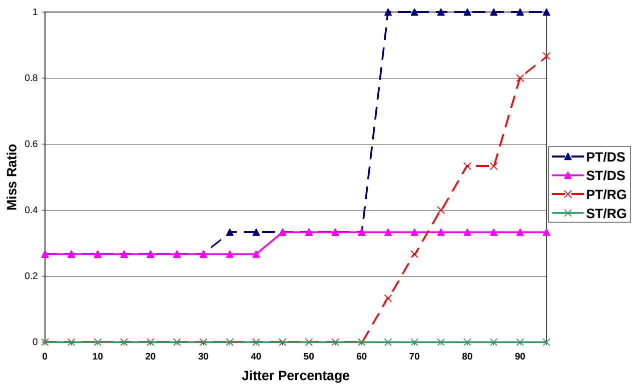

Figure 7. Miss ratio comparisons for 15 tasks when T_{10} is allowed x percent jitter. The time constraints for T_{10} are $\{(1,(1-x/100)*200),(2,400)\}$

To show the sharp contrast between the PT analysis and the ST analysis, we picked 75 percent jitter and calculated the WCRTs for all 15 tasks using four different algorithms. At that point, the time constraints for T_{10} are $\{(1,50),(2,400)\}$. In Figure 6, the WCRTs for all 15 tasks in PT/DS are infinite, and the WCRTs for six tasks in PT/RG are infinite, while the WCRTs for all tasks are bounded when using our generalized sporadic tasks analysis. In addition, DS performed much worse than RG for some low priority tasks. As shown in Figure 7, for the ST analysis with either the DS or the RG synchronization protocol, the miss ratios are fairly stable when the jitter percentage increases. In particular, the miss ratios are always 0 using the ST/RG algorithm. However, when the jitter percentage reaches 65 and the time constraints are

$\{(1, 70), (2, 400)\}$, all tasks are unschedulable using the PT/DS algorithm. Moreover, the PT/RG algorithm also has a steep rise in miss ratios when the jitter percentage is larger than 60.

To consider the influence of different time constraints on the miss ratios, we changed the time constraints of task T_{10} in two ways. In the first experiment whose results are shown in Figure 8, the number of arrivals allowed in the first time window is greater than one. It increases from 1 to 7 while the second time constraint ensures only 8 arrivals are allowed in any window of $8 * period$ length. In the second experiment whose results are shown in Figure 9, the number of time constraints for task T_{10} increases from 2 to 8.

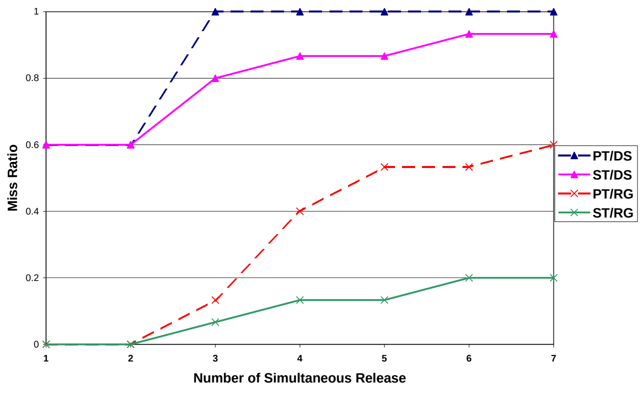

Figure 8. Miss ratio comparisons for 15 tasks when T_{10} is a generalized sporadic task with time constraints $\{(x, 200), (8, 1600)\}$

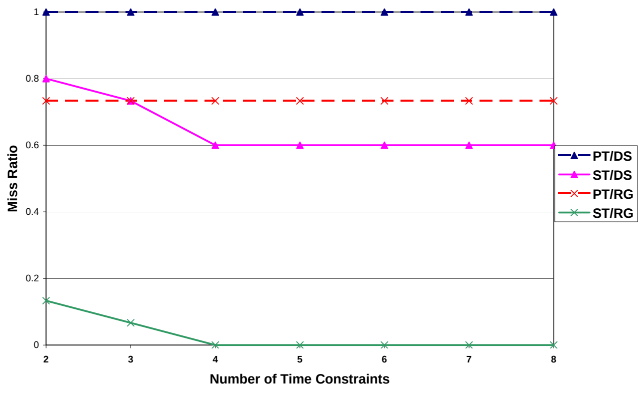

Figure 9. Miss ratio comparisons for 15 tasks when T_{10} is a generalized sporadic task. The number of time constraints for T_{10} increases from 2 to 8. The initial time constraints are $\{(1, 25), (8, 1600)\}$ at x=2 point. One extra time constraint $(3, 3/8 * 3 * 200)$ is added at x=3 point. At every x point, an extra time constraint $(x, x/8 * x * 200)$ is added.

As is shown in Figure 8, the PT analysis always had a higher miss ratio than our ST analysis and showed a sharp rise when the number of arrivals allowed in the first time window increased. In Figure 9, since the PT analysis only considers the first time contraint pair in its calculation, extra time constraints do not affect the miss ratio. However, our ST analysis considers all time constraints. Any tighter time constraint added may help reduce the miss ratio.

Besides the influence of different time constraints, we also considered the effect of the number of generalized sporadic tasks in the system. As is shown in Figure 10, when the number of generalized sporadic tasks in the system increases, the miss ratios under the PT/RG and PT/DS also increase, while the miss ratios are stable under the ST analysis and are lower than those under the PT analysis with the same synchronization protocol.

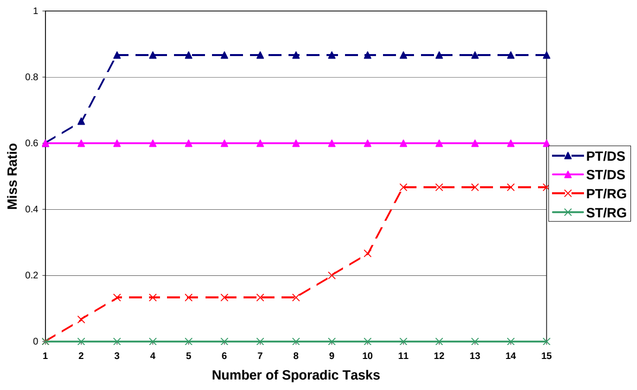

Figure 10. Miss ratio comparisons for 15 tasks when more and more periodic tasks are converted to generalized sporadic tasks with time constraints $\{(1, period/2), (2, period * 2)\}$. The number of generalized sporadic tasks increases from 1 to 15.

5 Related Work

The problem of scheduling a mixed set of hard periodic and soft aperiodic tasks in a dynamic environment has been widely considered when periodic tasks are executed under a fixed priority scheduling algorithm. Lehoczky et al. investigated bandwidth-preserving server mechanisms (Deferrable Server [13] and Priority Exchange [7]) to enhance aperiodic responsiveness. Sprunt et al. described a better service mechanism, called Sporadic Server (SS) [11]. Lehoczky and Ramos-Thuel found an optimal service method, called Slack Stealer [8], which is based on the idea of "stealing" all possible processing time from the periodic tasks without causing their deadlines to be missed. The same algorithm has been extended in [10] to handle hard aperiodic tasks, and in [2], to treat a more general class of scheduling problems. All these aperiodic servers require special scheduling mechanisms and nontrivial engineering effort on top of standard operating systems, while our analyses are based on the widely available fixed priority scheduling mechanism.

Although the sporadic task model [9] has been widely studied for supporting event-driven applications, event occurrences often cannot meet the assumption of tasks' minimum inter-arrival time. For example, tasks that are invoked in response to events generated by devices such as network interfaces may not satisfy this assumption. Rate-based scheduling schemes [3] are more seamlessly able to cope with jitter. In such schemes, there is no restriction on a task's instantaneous rate of execution, but an average rate is assumed. In multiprocessor systems, rate-based execution can be ensured by using scheduling algorithms that also ensure a property called proportionate fairness (Pfairness) [1]. In research on rate-based uniprocessor scheduling, Jeffay et al. [3, 5, 4] derived necessary and sufficient conditions for determining the feasibility of a rate-based task set and demonstrated that earliest deadline first (EDF) scheduling is optimal for both preemptive and non-preemptive execution environments. Baruah et al. [1] showed Pfair sheduling algorithms can be used to optimally schedule periodic tasks on multiprocessors. Srinivasan and Anderson [12] extended this work by showing that sporadic and rate-based tasks can also be optimally scheduled. Their sporadic task model is totally different from our generalized sporadic task model because the deadlines of the jobs of their sporadic tasks are not predefined, but assigned at the releasing times according to the tasks' average rates.

Wang et al. [15] presents a Priority-based Total Bandwidth Server (PTBS) to integrate the priority-driven scheduing paradigm with the share-driven scheduling paradigm for scheduling aperiodic tasks. Within each sliding window, the fixed priority is used to schedule different aperiodic or periodic jobs whose assigned deadlines fall into the window. Outside of the window, tasks are scheduled by EDF scheduling policy according to their assigned deadlines. The worst-case response time of aperiodic or periodic jobs had been derived and schedulability conditions provided when aperiodic tasks can be modeled by the leaky bucket arrival pattern. However, their schedulability condition only works in a single processor and cannot be easily extended to guarantee the schedulability of end-to-end aperiodic tasks. Moreover, our generalized sporadic task model has stronger descriptive capability than the leaky bucket model.

6 Conclusions

This paper has proposed a generalized sporadic task model that improves the traditional sporadic task model by characterizing arrival times more precisely. It also presented new schedulability analysis algorithms for generalized sporadic tasks, both for independent tasks and for end-to-end tasks synchronized either by direct synchronization or by a new generalized release guard synchronization protocol. Empirical results showed that our schedulability analyses, when compared with traditional analyses, (1) tighten the bounds on worst-case response times and (2) more effectively guarantee schedulability when sporadic tasks have greater arrival time jitter.

References

[1] S. K. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel. Proportionate progress: a notion of fairness in resource allocation. In *Proceedings of the ACM Symposium on the Theory of Computing*, 1993.

[2] R. I. Davis, K. Tindell, and A. Burns. Scheduling slack time in fixed priority preemptive systems. In *RTSS*, 1993.

[3] K. Jeffay and S. Goddard. A Theory of Rate-Based Execution. In *RTSS*, 1999.

[4] K. Jeffay and S. Goddard. Rate-Based Resource Allocation Methods for Embedded Systems. In *EMSOFT*, 2001.

[5] K. Jeffay and G. Lamastra. A Comparative Study of the Realization of Rate-Based Computing Services in General Purpose Operating Systems. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications*, 2000.

[6] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *RTSS*, 1990.

[7] J. P. Lehoczky, L. Sha, and J. K. Strosnider. Enhanced aperiodic responsiveness in a hard real-time environment. In *RTSS*, 1987.

[8] J. P. Lehoczky and S. R. Thuel. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *RTSS*, 1992.

[9] A. K. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. PhD thesis, Massachusetts Institute of Technology, 1983.

[10] S. Ramos-Thuel and J. P. Lehoczky. On-line scheduling of hard deadline aperiodic tasks in fixed-prioriry systems. In *RTSS*, 1993.

[11] B. Sprunt, L. Sha, and L. Lehoczky. Aperiodic task scheduling for hard real-time systems. *The Journal of Real-Time Systems*, 1(1):27–60, 1989.

[12] A. Srinivasan and J. Anderson. Optimal Rate-based Scheduling on Multiprocessors. In *Proceedings of the 34th ACM Symposium on Theory of Computing*, 2002.

[13] J. Strosnider, J. P. Lehoczky, and L. Sha. The deferrable server algorithm for enhanced aperiodic responsiveness in real-time environments. *IEEE Transactions on Computers*, 44(1):73–91, 1995.

[14] J. Sun. *Fixed-Priority End-to-End scheduling in Distributed Real-Time Systems*. PhD thesis, Department of Computer Science, UIUC, 1997.

[15] S. Wang, Y. Wang, and K. Lin. Integrating the Fixed Priority Scheduling and the Total Bandwidth Server for Aperiodic Tasks. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications*, 2000.