Task Scheduling with Self-Suspensions in Soft Real-Time Multiprocessor Systems*

Cong Liu and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill

Abstract

In work on multiprocessor real-time systems, task scheduling with self-suspensions is a relatively unexplored topic. In this paper, soft real-time sporadic task systems are considered that include self-suspending tasks. Conditions are presented for guaranteeing bounded deadline tardiness in such systems under global EDF or FIFO multiprocessor scheduling. These conditions enable many soft real-time task systems with self-suspending tasks to be scheduled with little or no utilization loss.

1 Introduction

In many real-time systems, tasks interact with external devices that introduce self-suspension delays. Examples of such devices include solid-state and magnetic disks and network cards. Delays introduced by such devices can be moderate (e.g., roughly $15\mu s$ per read and $200\mu s$ per write for NAND Flash) or quite lengthy (e.g., roughly $15ms$ for magnetic disks) [3, 5]. Unfortunately, such delays quite negatively impact schedulability in real-time systems if deadline misses cannot be tolerated [12].

In this paper, we consider whether, on multiprocessor platforms, such negative impacts can be ameliorated if task deadlines are soft. Our focus on multiprocessors is motivated by the advent of multicore platforms. There is currently great interest in providing operating-system support to enable real-time workloads to be hosted on such platforms. Many such workloads can be expected to include self-suspending tasks. Moreover, in many settings, such workloads can be expected to have soft timing constraints. The soft timing constraint considered in this paper pertains to implicit-deadline sporadic task systems and requires that deadline tardiness be bounded.

All multiprocessor scheduling algorithms follow either a *partitioning* or *global-scheduling* approach (or some combination of the two). Under partitioning, tasks are statically assigned to processors, while under global scheduling, they may migrate. Under partitioning schemes, constraints on overall utilization are required to ensure timeliness even if bounded deadline tardiness can be tolerated. On the other hand, a variety of global-scheduling approaches are capable of ensuring bounded tardiness in sporadic systems (without self-suspending tasks), as long as the system is not over-utilized [6].

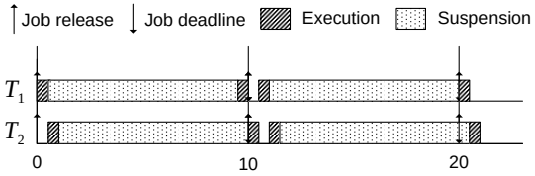

Figure 1: Example task system with low utilization.

Unfortunately, if tasks may self-suspend, then bounded tardiness cannot be guaranteed even on uniprocessors without constraining the system in some way. Consider, for example, the uniprocessor task system, scheduled by the earliest-deadline-first (EDF) algorithm, shown in Fig. 1. This system consists of two tasks, each of which releases a new job every 10 time units that executes for 0.5 time unit, then suspends for 9 time units, and then executes for another 0.5 time unit. Tardiness grows unboundedly in this system, even though its total utilization is small.

Motivated by the observations above, we consider in this paper whether it is possible to specify reasonable constraints under which bounded deadline tardiness is guaranteed under global scheduling algorithms, for implicit-deadline sporadic tasks systems with self-suspending tasks. We focus specifically on two global scheduling algorithms that are capable of ensuring bounded tardiness for ordinary (suspension-less) sporadic systems with no utilization loss [2, 7], namely, the *global earliest-deadline-first* (GEDF) algorithm and the *global first-in first-out* (GFIFO) algorithm.

Related work. To our knowledge, self-suspensions have not been considered before in the context of global real-time scheduling algorithms.

In work pertaining to uniprocessors (and by extension multiprocessors scheduled via partitioning), several schedulability tests have been presented for analyzing tasks with self-suspensions. These include a utilization-based test for

*Work supported by AT&T, IBM, Intel, and Sun Corps.; NSF grants CNS 0834270, CNS 0834132, and CNS 0615197; and ARO grant W911NF-06-1-0425.

EDF [1] and response-time-bound tests for fixed-priority systems [4, 8–10] and EDF-scheduled systems [10]. On a more negative note, Ridouard et. al [13] have shown that the feasibility problem for hard real-time, independent tasks with self-suspensions is NP-hard in the strong sense. Ridouard and Richard [12] have also shown that no optimal on-line algorithm exists for task systems with self-suspensions. In fixed-priority systems, scheduling penalties associated with self-suspensions can be lessened by using a technique called the *period enforcer* [11], which forces suspensions to occur more predictably.

Contributions. We show that GEDF's and GFIFO's ability to guarantee bounded tardiness in sporadic systems with self-suspending tasks hinges upon a task parameter that we call the "maximum suspension ratio," denoted ξ^H_{max}, with range [0,1]. We present a general tardiness bound, which is applicable to either GEDF or GFIFO, that expresses tardiness as a function of ξ^H_{max} and other task parameters. This bound shows that task systems consisting of both self-suspending tasks and ordinary computational tasks that do not suspend can be supported with bounded tardiness if $\xi^H_{max} < 1 - \frac{U^s_{sum} + U^c_L}{m}$, where m is the number of processors, U^s_{sum} is the total utilization of self-suspending tasks, and U^c_L is the total utilization of the $m-1$ computational tasks of highest utilization. The task model assumed in obtaining this result is very general and allows self-suspensions within a task's jobs to interleave arbitrarily with computation. We show via a counterexample that task systems that violate our utilization constraint may have unbounded tardiness. To assess the impact of this constraint, we present schedulability experiments that compare our analysis to a common approach for analyzing systems with self-suspensions wherein suspensions are merely treated as computation. In these experiments, our approach proved to be superior (i.e., could guarantee bounded tardiness for more systems) in most of the tested scenarios.

Organization. The rest of this paper is organized as follows. Sec. 2 describes our system model. In Sec. 3, our tardiness bound is derived and evaluated. Sec. 4 concludes.

2 System Model

We consider the problem of scheduling a set $\tau = \{T_1, ..., T_n\}$ of n independent sporadic tasks on $m \geq 2$ identical processors. Each task is released repeatedly, with each such invocation called a *job*. Jobs alternate between computation and suspension phases. We assume that each job of T_l executes for at most e_l time units (across all of its execution phases) and suspends for at most s_l time units (across all of its suspension phases). We place no restrictions on how these phases interleave (a job can even begin or end with a suspension phase). The j^{th} job of T_l, denoted $T_{l,j}$, is released at time $r_{l,j}$ and has a deadline at time $d_{l,j}$. Associated with each task T_l is a period p_l, which specifies both the minimum time between two consecutive job releases of T_l and the relative deadline of each such job, i.e., $d_{l,j} = r_{l,j} + p_l$. The utilization of a task T_l is defined as $u_l = e_l/p_l$, and the utilization of the task system τ as $U_{sum} = \sum_{T_i \in \tau} u_i$. We require $e_l + s_l \leq p_l$, $u_l \leq 1$, and $U_{sum} \leq m$; otherwise, tardiness can grow unboundedly (if $e_l + s_l > p_l$ or $u_l > 1$, then T_l's tardiness grows unboundedly; if $U_{sum} > m$, then the system is overloaded, which implies that tardiness grows unboundedly for at least one task).

A common case for real-time workloads is that both self-suspending tasks and computational tasks (which do not suspend) co-exist. To reflect this, we let U^s_{sum} denote the total utilization of all self-suspending tasks, and U^c_{sum} denote the total utilization of all computational tasks.

Successive jobs of the same task are required to execute in sequence. If a job $T_{i,j}$ completes at time t, then its *response time* is $t - r_{i,j}$ and its *tardiness* is $max(0, t - d_{i,j})$. A task's tardiness is the maximum tardiness of any of its jobs. Note that, when a job of a task misses its deadline, the release time of the next job of that task is not altered.

Unless stated otherwise, we henceforth assume that each job of any task T_l executes for *exactly* e_l time units. By Claim 4, given in an appendix, any tardiness bound derived for systems that meet this restriction applies to other systems as well.

So that our analysis can be more accurately applied in settings where a task's total suspension time varies from job to job, we assume that a fixed parameter H ($H \geq 1$) is specified and that S^H_i denotes the maximum total self-suspension length for any H ($H \geq 1$) consecutive jobs of task T_i. Note that if $H = 1$, then a maximum per-job total suspension length is being assumed.

Under GEDF (GFIFO), released jobs are prioritized by their deadlines (release times). So that our results can be applied to both algorithms, we consider a generic scheduling algorithm (GSA) where each job is prioritized by some time point between its release time and deadline. Specifically, for any job $T_{i,j}$, we define a priority value $\rho_{i,j} = r_{i,j} + \kappa \cdot p_i$, where $0 \leq \kappa \leq 1$. Lower priority values denote higher priorities, and ties are assumed to be broken in favor of tasks with smaller indices. Note that GEDF and GFIFO are special cases of GSA where κ is set to 1 and 0, respectively.

3 A Tardiness Bound for GSA

We derive a tardiness bound for GSA by comparing the allocations to a task system τ in a processor sharing (PS) schedule and an actual GSA schedule of interest for τ, both on m

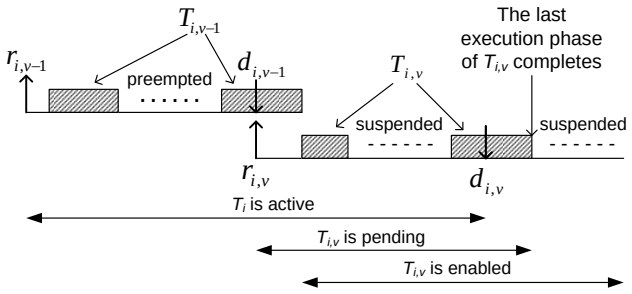

Figure 2: Illustration of Defs. 1-3.

processors, and quantifying the difference between the two. We analyze task allocations on a per-task basis.

The time interval $[t_1, t_2)$, where $t_2 > t_1$, consists of all time instances t, where $t_1 \leq t < t_2$, and is of length $t_2 - t_1$. For any time $t > 0$, the notation t^- is used to denote the time $t - \varepsilon$ in the limit $\varepsilon \to 0+$, and the notation t^+ is used to denote the time $t + \varepsilon$ in the limit $\varepsilon \to 0+$.

Definition 1. A task T_i is *active* at time t if there exists a job $T_{i,v}$ such that $r_{i,v} \leq t < d_{i,v}$.

Definition 2. Job $T_{i,v}$ is *pending* at time t if $t \geq r_{i,v}$ and $T_{i,v}$ has not completed all of its execution phases by t. Note that $T_{i,v}$ is not pending at t if it has completed all its execution phases by t but not all of its suspension phases.

Definition 3. Job $T_{i,v}$ is *enabled* at t if $t \geq r_{i,v}$, $T_{i,v}$ has not completed by t, and its predecessor (if any) has completed by t. A job is considered to be completed if it has finished its last phase (be it suspension or computation).

The above three definitions are illustrated in Fig. 2.

Let $A(T_{i,j}, t_1, t_2, S)$ denote the total allocation to the job $T_{i,j}$ in an arbitrary schedule S in $[t_1, t_2)$. Then, the total time allocated to all jobs of T_i in $[t_1, t_2)$ in S is given by

$$A(T_i, t_1, t_2, S) = \sum_{j \geq 1} A(T_{i,j}, t_1, t_2, S).$$

Consider a PS schedule *PS*. In such a schedule, T_i executes with the rate u_i when it is active. (*Note that suspensions are not considered in the PS schedule*.) Thus, if T_i is active throughout $[t_1, t_2)$, then

$$A(T_{i,j}, t_1, t_2, PS) = (t_2 - t_1)u_i. \quad (1)$$

The difference between the allocation to a job $T_{i,j}$ up to time t in a PS schedule and an arbitrary schedule S, denoted *the lag of job $T_{i,j}$ at time t in schedule S*, is defined by

$$lag(T_{i,j}, t, S) = A(T_{i,j}, 0, t, PS) - A(T_{i,j}, 0, t, S). \quad (2)$$

The concept of lag is important because, if lags remain bounded, then tardiness is bounded as well. The *LAG* for a finite job set J at time t in the schedule S is defined by

$$LAG(J, t, S) = \sum_{T_{i,j} \in J} lag(T_{i,j}, t, S)$$
$$= \sum_{T_{i,j} \in J} (A(T_{i,j}, 0, t, PS) - A(T_{i,j}, 0, t, S)). \quad (3)$$

Our tardiness-bound derivation follows a format originally presented in [2] and focuses on a given task system τ. We order the jobs in τ based on their priorities: $T_{i,v} \prec T_{a,b}$ iff $\rho_{i,v} < \rho_{a,b}$ or $(\rho_{i,v} = \rho_{a,b}) \wedge (i < a)$. Let $T_{l,j}$ be a job of a task T_l in τ, $t_d = d_{l,j}$, and S be a GSA schedule for τ with the following property.

(P) The tardiness of every job $T_{i,k}$ such that $T_{i,k} \prec T_{l,j}$ is at most $x + e_i + s_i$ in S, where $x \geq 0$.

Our objective is to determine the smallest x such that the tardiness of $T_{l,j}$ is at most $x + e_l + s_l$. This would by induction imply a tardiness of at most $x + e_i + s_i$ for all jobs of every task T_i, where $T_i \in \tau$. We assume that $T_{l,j}$ finishes after t_d, for otherwise, its tardiness is trivially zero. The steps for determining the value for x are as follows.

1. Determine an upper bound on the work pending for tasks in τ that can compete with $T_{l,j}$ after t_d. This is dealt with in Lemmas 1–3 in Sec. 3.1.

2. Determine a lower bound on the amount of work pending for tasks in τ that can compete with $T_{l,j}$ after t_d, required for the tardiness of $T_{l,j}$ to exceed $x + e_l + s_l$. This is dealt with in Lemma 4 in Sec. 3.2.

3. Determine the smallest x such that the tardiness of $T_{l,j}$ is at most $x + e_l + s_l$, using the above upper and lower bounds. This is dealt with in Theorem 1 in Sec. 3.3.

Definition 4. We categorize jobs based on the relationship between their priorities and deadlines and those of $T_{l,j}$:

$$\mathbf{d} = \{T_{i,v} : (T_{i,v} \preceq T_{l,j}) \wedge (d_{i,v} \leq t_d)\};$$

$$\mathbf{D} = \{T_{i,v} : (T_{i,v} \prec T_{l,j}) \wedge (d_{i,v} > t_d)\}.$$

$\mathbf{d}$ is the set of jobs with deadlines at most t_d with priority at least that of $T_{l,j}$. These jobs do not execute beyond t_d in the PS schedule. Note that $T_{l,j}$ is in $\mathbf{d}$. $\mathbf{D}$ is the set of jobs that have higher priorities than $T_{l,j}$ and deadlines greater than t_d. Note that jobs not in $\mathbf{d} \cup \mathbf{D}$ have lower priority than those in $\mathbf{d} \cup \mathbf{D}$ and thus do not affect the scheduling of jobs in $\mathbf{d} \cup \mathbf{D}$. For simplicity, we will henceforth assume that no job not in $\mathbf{d} \cup \mathbf{D}$ executes in either the PS or GSA schedule. Let D_{CI} be the set of tasks with jobs in $\mathbf{D}$. $\mathbf{D}$ consists of *carry-in jobs*, which have a release time before t_d and a deadline after t_d. Exactly one such job exists for each task in D_{CI}. (Note that $\mathbf{D}$ is empty under GEDF because jobs with later deadlines have lower priorities.)

Definition 5. A time instant t is *busy* for a job set J if all m processors execute a job in J at t. A time interval is *busy* for J if each instant within it is busy for J.

The following claim follows from the definition of LAG.

Claim 1. *If $LAG(\boldsymbol{d}, t_2, S) > LAG(\boldsymbol{d}, t_1, S)$, where $t_2 > t_1$, then $[t_1, t_2)$ is non-busy for **d**. In other words, LAG for **d** can increase only throughout a non-busy interval.*

An interval could be non-busy for **d** for two reasons:

1. There are not enough enabled non-suspended jobs in **d** to occupy all available processors. Such an interval is called *non-busy non-displacing*.
2. There are enabled non-suspended jobs in **d** that are not scheduled (because jobs in **D** occupy one or more processors).

Definition 6. Let δ_k be the amount of execution time consumed by a carry-in job $T_{k,v}$ by time t_d.

Definition 7. Let $B(\mathbf{D}, t_d, S)$ be the amount of work due to jobs in **D** that can compete with $T_{l,j}$ after t_d.

Since $\mathbf{d} \cup \mathbf{D}$ includes all jobs of higher priority than $T_{l,j}$, the competing work for $T_{l,j}$ is given by the sum of **(i)** the amount of work pending at t_d for jobs in **d**, and **(ii)** the amount of work $B(\mathbf{D}, t_d, S)$ demanded by jobs in **D** that competes with $T_{l,j}$ after t_d. Since jobs from **d** have deadlines at most t_d, they do not execute in the PS schedule beyond t_d. Thus, the work pending for them is given by $LAG(\mathbf{d}, t_d, S)$. Therefore, the competing work for $T_{l,j}$ after t_d is given by $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S)$. Let

$$Z = LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S). \tag{4}$$

A summary of the terms defined so far, as well as some additional terms defined later, is presented in Table 1.

3.1 Upper Bound

In this section, we determine an upper bound on Z.

Definition 8. Let t_n be the end of the latest non-busy non-displacing interval for **d** before t_d, if any; otherwise, $t_n = 0$.

The following two lemmas have been proved previously for both GEDF [2] and GFIFO [7] for ordinary sporadic task systems without self-suspensions. Note that the value of $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S)$ depends only on allocations in the PS schedule PS and allocations to jobs in $\mathbf{d} \cup \mathbf{D}$ in the actual schedule S by time t_d. The PS schedule is not impacted by self-suspensions. Also, Property (P) alone is sufficient for determining how much work any job in $\mathbf{d} \cup \mathbf{D}$ other than $T_{l,j}$ completes before t_d. For these reasons, Lemmas 1 and 2 continue to hold for task systems with self-suspensions. For completeness, proofs are given in the appendix.

m	Number of processors
n	Number of tasks
$S(T_{l,j})$	Start time of job $T_{l,j}$
$F(T_{l,j})$	Finish time of job $T_{l,j}$
t_d	Deadline of job $T_{l,j}$
E^s_{sum}	Total execution cost of all self-suspending tasks in τ
E_{sum}	Total execution cost of all tasks in τ
S^s_{sum}	Total suspension length of all tasks in τ
u^s_{max}	Maximum utilization of any self-suspending task in τ
U^c_L	Sum of the $\min(m-1, c)$ largest computational task utilizations, where c is the number of computational tasks
E^c_L	Sum of the $\min(m-1, c)$ largest computational task execution costs
S^H_i	Maximum total self-suspension length for any H ($H > 1$) consecutive jobs of task T_i
S^H_{max}	Maximum total self-suspension length for any H ($H > 1$) consecutive jobs of any task in τ
ξ^H_i	Suspension ratio of T_i
ξ^H_{max}	Maximum suspension ratio
δ_k	Amount of execution time consumed by a carry-in job $T_{k,v}$ by time t_d
$B(\mathbf{D}, t_d, S)$	Amount of work due to jobs in **D** that can compete with $T_{l,j}$ after t_d
W	Amount of work due to jobs in $\mathbf{d} \cup \mathbf{D}$ that can compete with $T_{l,j}$ at or after $t_d + y$, including the work due for $T_{l,j}$
Z	Amount of competing work for $T_{l,j}$ after t_d
t_s	Earliest non-busy instant in $[t_d, t_d + y)$
t_p	Finish time of $T_{l,j}$'s predecessor, if it exists; otherwise ($j = 1$), $t_p = 0$

Table 1: Summary of notation.

Lemma 1. *$LAG(\boldsymbol{d}, t_d, S) \leq LAG(\boldsymbol{d}, t_n, S) + \sum_{T_k \in D_{CI}} \delta_k(1 - u_k)$, where $t \in [0, t_d]$.*

Lemma 2. *$lag(T_i, t, S) \leq u_i \cdot x + e_i + u_i \cdot s_i$ for any task T_i and $t \in [0, t_d]$.*

Lemma 3 below upper bounds $LAG(\mathbf{d}, t_n, S)$.

Definition 9. Let E^s_{sum} be the total execution cost of all self-suspending tasks in τ. Let E_{sum} be the total execution cost of all tasks in τ. Let S^s_{sum} be the total suspension length of all tasks in τ. Let u^s_{max} be the maximum utilization of any self-suspending task in τ.

Definition 10. Let U^c_L be the sum of the $\min(m-1, c)$ largest computational task utilizations, where c is the num-

ber of computational tasks. Let E_L^c be the sum of the $\min(m-1, c)$ largest computational task execution costs.

Lemma 3. $LAG(\boldsymbol{d}, t_n, S) \leq (U_{sum}^s + U_L^c) \cdot x + E_{sum}^s + E_L^c + u_{max}^s \cdot S_{sum}^s$.

Proof. By summing individual task lags at t_n, we can bound $LAG(\mathbf{d}, t_n, S)$. If $t_n = 0$, then $LAG(\mathbf{d}, t_n, S) = 0$, so assume $t_n > 0$. Consider the set of tasks $\beta = \{T_i : \exists T_{i,v}$ in $\mathbf{d}$ such that $T_{i,v}$ is enabled at $t_n^-\}$. Given that the instant t_n^- is non-busy non-displacing, at most $m-1$ computational tasks in β have jobs executing at t_n^-. Due to suspensions, however, β may contain more than $m-1$ tasks. In the worst case, all suspending tasks in τ have a suspended enabled job at t_n^- and $\min(m-1, c)$ computational tasks have an enabled job executing at t_n^-. If task T_i does not have an enabled job at t_n^-, then $lag(T_i, t_n, S) \leq 0$. Therefore, by (3), we have

$$\begin{aligned} LAG(\mathbf{d}, t_n, S) &= \sum_{T_i : T_{i,v}^w \in \mathbf{d}} lag(T_i, t_n, S) \\ &\leq \sum_{T_i \in \beta} lag(T_i, t_n, S) \\ &\quad \{\text{by Lemma 2}\} \\ &\leq \sum_{T_i \in \beta} (u_i \cdot x + e_i + u_i \cdot s_i) \\ &\leq (U_{sum}^s + U_L^c) \cdot x + E_{sum}^s + E_L^c \\ &\quad + u_{max}^s \cdot S_{sum}^s. \end{aligned}$$ □

The demand placed by jobs in $\mathbf{D}$ after t_d is $B(\mathbf{D}, t_d, S) = \sum_{T_k \in D_{CI}} (e_k - \delta_k)$. Thus, by (4) and Lemmas 1 and 3, we have the following upper bound:

$$\begin{aligned} Z &\leq (U_{sum}^s + U_L^c) \cdot x + E_{sum}^s + E_L^c + u_{max}^s \cdot S_{sum}^s \\ &\quad + \sum_{T_k \in D_{CI}} (\delta_i(1-u_k) + (e_k - \delta_k)) \\ &\leq (U_{sum}^s + U_L^c) \cdot x + E_{sum}^s + E_L^c + u_{max}^s \cdot S_{sum}^s \\ &\quad + E_{sum}. \end{aligned} \tag{5}$$

3.2 Lower Bound

Lemma 4, given below, establishes a lower bound on Z that is necessary for the tardiness of $T_{l,j}$ to exceed $x + e_l + s_l$.

Definition 11. If job $T_{i,v}$ is enabled and not suspended at time t but does not execute at t, then it is *preempted* at t.

Definition 12. If $T_{i,v}$'s first phase is an execution (suspension) phase and it begins executing (a suspension) for the first time at t, then t is called its *start time*, denoted $S(T_{i,v})$. If $T_{i,v}$'s last phase (be it execution or suspension) completes at time t', then t' is called its *finish time*, denoted $F(T_{i,v})$.

Definition 13. Let $S_{max}^H = \max\{S_1^H, S_2^H, ..., S_n^H\}$. ($S_i^H$ was defined earlier in Sec. 2.)

Definition 14. Let $\xi_i^H = \dfrac{S_{max}^H}{S_{max}^H + H \cdot e_i}$ be the *suspension ratio* of T_i. Let $\xi_{max}^H = \max\{\xi_1, \xi_2, ..., \xi_n\}$ be the *maximum suspension ratio*.

Lemma 4. *If the tardiness of $T_{l,j}$ exceeds $x + e_l + s_l$, then* $Z > (1-\xi_{max}^H) \cdot mx - (m-1)e_l - m \cdot s_l - n \cdot (S_{max}^H + 2S_{max}^1)$.

Proof. We prove the contrapositive: we assume that

$$\begin{aligned} Z &\leq (1 - \xi_{max}^H) \cdot mx - (m-1)e_l \\ &\quad - m \cdot s_l - n \cdot (S_{max}^H + 2S_{max}^1) \end{aligned} \tag{6}$$

holds and show that the tardiness of $T_{l,j}$ cannot exceed $x + e_l + s_l$. Let η_l be the amount of work $T_{l,j}$ performs by time t_d in S. Define y as follows.

$$y = (1 - \xi_{max}^H) \cdot x + \frac{\eta_l}{m} \tag{7}$$

Let W be the amount of work due to jobs in $\mathbf{d} \cup \mathbf{D}$ that can compete with $T_{l,j}$ at or after $t_d + y$, including the work due for $T_{l,j}$. We consider two cases.

Case 1. $[t_d, t_d + y)$ *is a busy interval for* $\mathbf{d} \cup \mathbf{D}$. In this case, the amount of work due to jobs in $\mathbf{d} \cup \mathbf{D}$ performed within $[t_d, t_d + y]$ is my, and hence, $W = Z - my$. Thus, by (6) and (7), $W \leq (1-\xi_{max}^H) \cdot mx - (m-1)e_l - m \cdot s_l - n \cdot (S_{max}^H + 2S_{max}^1) - my = (1-\xi_{max}^H) \cdot mx - (m-1)e_l - m \cdot s_l - n \cdot (S_{max}^H + 2S_{max}^1) - (1-\xi_{max}^H) \cdot mx - \eta_l < 0$. Since $T_{l,j}$ can suspend for at most s_l time units after $t_d + y$ (and at least one task executes while it is not suspended), the amount of work performed by the system for jobs in $\mathbf{d} \cup \mathbf{D}$ during the interval $[t_d + y, F(T_{l,j}))$ is at least $F(T_{l,j}) - t_d - y - s_l$. Hence, $F(T_{l,j}) - t_d - y - s_l \leq W < 0$. Therefore, the tardiness of $T_{l,j}$ is $F(T_{l,j}) - t_d < y + s_l = (1 - \xi_{max}^H) \cdot x + \dfrac{\eta_l}{m} + s_l \leq x + e_l + s_l$.

Case 2. $[t_d, t_d + y)$ *is a non-busy interval for* $\mathbf{d} \cup \mathbf{D}$. Let $t_s \geq t_d$ be the earliest non-busy instant in $[t_d, t_d + y)$. Job $T_{l,j}$ cannot become enabled until its predecessor (if it exists) completes. Let t_p be the finish time of $T_{l,j}$'s predecessor (i.e., $T_{i,j-1}$), if it exists; otherwise ($j = 1$), let $t_p = 0$. We consider three subcases.

Subcase 2.1. $t_p \leq t_s$ *and* $T_{l,j}$ *is not preempted after* t_s. In this case, $T_{l,j}$ performs its remaining execution and suspension phases in sequence without preemption after t_s (note that, by Def. 11, $T_{l,j}$ is not considered to be preempted when it is suspended). Thus, because $t_s < t_d + y$, by (7), the tardiness of $T_{l,j}$ is at most $t_s + e_l - \eta_l + s_l - t_d < t_d + y + e_l - \eta_l + s_l - t_d = (1-\xi_{max}^H) \cdot x + \dfrac{\eta_l}{m} + e_l - \eta_l + s_l \leq x + e_l + s_l$.

The claim below will be used in the next two subcases.

Claim 2. *The amount of work due to* $\boldsymbol{d} \cup \boldsymbol{D}$ *performed within $[t_1, t_2)$, where $S(T_{l,j}) \leq t_1 < t_2 \leq F(T_{l,j})$, is at least* $m(t_2 - t_1) - (m-1)e_l - m \cdot s_l$.

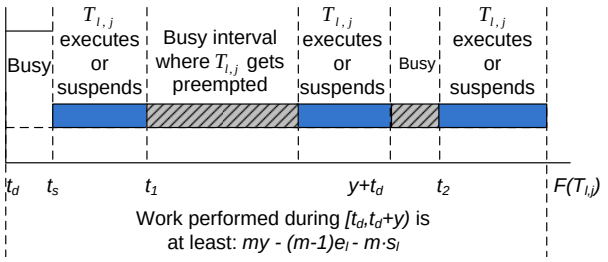

Figure 3: Subcase 2.2.

Proof. Within $[t_1, t_2)$, all intervals during which $T_{l,j}$ is preempted are busy, and $T_{l,j}$ can execute for at most e_l time. Within intervals where $T_{l,j}$ executes, at least one processor is occupied by $T_{l,j}$. Thus, at most $m-1$ processors are idle while $T_{l,j}$ executes (for at most e_l time units) in $[t_1, t_2)$. Also, all processors can be idle while $T_{l,j}$ is suspended and this happens for at most s_l time units in $[t_1, t_2)$. □

Subcase 2.2. *$t_p \leq t_s$ and $T_{l,j}$ is preempted after t_s.* Let t_1 be the earliest time when $T_{l,j}$ is preempted after t_s, and let t_2 be the last time $T_{l,j}$ resumes execution after being preempted. (A finite number of jobs have higher priority than $T_{l,j}$, so t_2 exists.) Then, as shown in Fig. 3, $T_{l,j}$ executes or suspends within $[t_s, t_1)$. Also, because $T_{l,j}$ is preempted at t_1, t_1 is busy with respect to $\mathbf{d} \cup \mathbf{D}$. Within $[t_1, t_2)$, $T_{l,j}$ could be repeatedly preempted. All such intervals during which $T_{l,j}$ is preempted must be busy in order for the preemption to happen. Note that $F(T_{l,j}) \leq t_2 + e_l - \eta_l + s_l$. Thus, if $t_2 \leq y + t_d$, then $F(T_{l,j}) \leq y + t_d + e_l - \eta_l + s_l$, which by (7) implies that $T_{l,j}$'s tardiness is $F(T_{l,j}) - t_d \leq y + e_l - \eta_l + s_l \leq (1 - \xi^H_{max}) \cdot x + e_l + s_l \leq x + e_l + s_l$, as required. If $t_2 > t_d + y$, then by Claim 2, the amount of work due to $\mathbf{d} \cup \mathbf{D}$ performed within $[t_s, t_d + y)$ is at least $m(t_d + y - t_s) - (m-1)e_l - m \cdot s_l$. Because $[t_d, t_s)$ is busy, the work due to $\mathbf{d} \cup \mathbf{D}$ performed within $[t_d, t_d + y)$ is thus at least $my - (m-1)e_l - m \cdot s_l$. Hence, the amount of work that can compete with $T_{l,j}$ (including work due to $T_{l,j}$) at or after $t_d + y$ is

$$\begin{aligned} W &\leq Z - (my - (m-1)e_l - m \cdot s_l) \\ &\quad \{\text{by (6)}\} \\ &\leq (1 - \xi^H_{max}) \cdot mx - (m-1)e_l - m \cdot s_l - \\ &\quad n \cdot (S^H_{max} + 2S^1_{max}) - (my - (m-1)e_l - m \cdot s_l) \\ &= (1 - \xi^H_{max}) \cdot mx - n \cdot (S^H_{max} + 2S^1_{max}) - my \\ &\quad \{\text{by (7)}\} \\ &= -n \cdot (S^H_{max} + 2S^1_{max}) - \eta_l \\ &\leq 0. \end{aligned}$$

Therefore, the tardiness of $T_{l,j}$ is $F(T_{l,j}) - t_d \leq y + W \leq y = (1 - \xi^H_{max}) \cdot x + \dfrac{\eta_l}{m} < x + e_l + s_l$.

Subcase 2.3: $t_p > t_s$. The earliest time $T_{l,j}$ can commence its first phase (be it an execution or suspension phase) is t_p, as shown in Fig. 4. If fewer than m tasks have enabled jobs in $\mathbf{d} \cup \mathbf{D}$ at any time instant within $[t_s, t_p)$, then $T_{l,j}$ will begin its first phase at t_p and finish by time $t_p + e_l + s_l$. (Note that the number of enabled jobs in $\mathbf{d} \cup \mathbf{D}$ does not increase after t_d.) By Property (P) (applied to $T_{l,j}$'s predecessor), $t_p \leq t_d - p_l + x + e_l + s_l \leq t_d + x$. Thus, the tardiness of $T_{l,j}$ is $F(T_{l,j}) - t_d \leq t_p + e_l + s_l - t_d \leq x + e_l + s_l$.

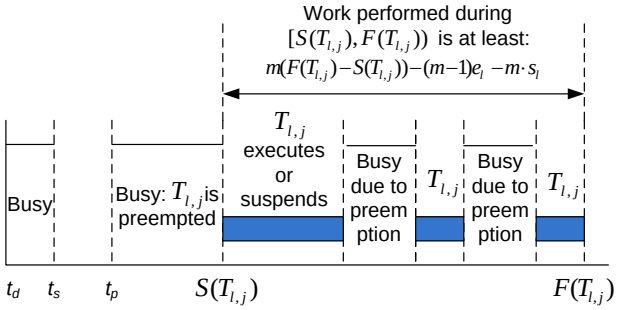

Figure 4: Subcase 2.3.

The remaining possibility (which requires a much lengthier argument) is: $t_p > t_s$ and at least m tasks have enabled jobs in $\mathbf{d} \cup \mathbf{D}$ at each time instant within $[t_s, t_p)$. In this case, given that at least m tasks have enabled jobs in $\mathbf{d} \cup \mathbf{D}$ at t_s, t_s is non-busy due to suspensions.

Let W' be the amount of work due to $\mathbf{d} \cup \mathbf{D}$ performed during $[t_s, t_p)$. Let I be the total idle time in $[t_s, t_p)$, where the idle time at each instant is the number of idle processors at that instant. Then, $W' + I = m \cdot (t_p - t_s)$. The following claim will be used to complete the proof of Subcase 2.3.

Claim 3. $W' \geq (1 - \xi^H_{max}) \cdot m(t_p - t_s) - n \cdot (S^H_{max} + 2S^1_{max})$.

Proof. We begin by dividing the interval $[t_s, t_p)$ into subintervals on a per-processor basis. The subintervals on processor k are denoted $[IT^{(k)}_i, ET^{(k)}_i)$, where $1 \leq i \leq q_k$, $IT^{(k)}_i = t_s$, $IT^{(k)}_{i+1} = ET^{(k)}_i$, and $ET^{(k)}_{q_k} = t_p$, as illustrated in Fig 5. With each such subinterval $[IT^{(k)}_i, ET^{(k)}_i)$, we associate a unique task, denoted $T^{(k)}_i$. We assume that during $[IT^{(k)}_i, ET^{(k)}_i)$, $T^{(k)}_i$ executes only on processor k, and $ET^{(k)-}_i$ is the last time $T^{(k)}_i$ is enabled within $[t_s, t_p)$. Thus, if $ET^{(k)}_i < t_p$, then the last job of $T^{(k)}_i$ to be enabled within $[t_s, t_p)$ finishes its last phase (be it execution or suspension) at time $ET^{(k)-}_i$; if $ET^{(k)}_i = t_p$, then $T^{(k)}_i$ has enabled jobs throughout $[IT^{(k)}_i, t_p)$. Note that it is possible that $T^{(k)}_i$ executes or suspends within $[t_s, t_p)$ prior to $IT^{(k)}_i$. We call the subinterval $[IT^{(k)}_i, ET^{(k)}_i)$ the *presence interval* of $T^{(k)}_i$. The fact that a unique task can be associated with each subinterval follows from the assumption that at least m tasks have jobs in $\mathbf{d} \cup \mathbf{D}$ that are enabled

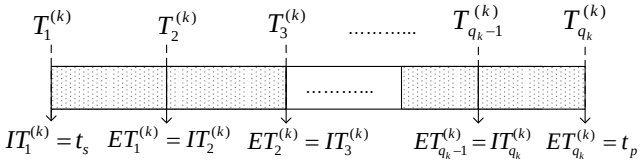

Figure 5: Presence intervals within $[t_s, t_p]$.

at each time instant in $[t_s, t_p)$.[1] (Note that multiple jobs of $T_i^{(k)}$ may execute during its presence interval.) We let $\lambda^{(k)}$ denote the set of all tasks that have presence intervals on processor k.

We now upper-bound the idleness on processor k by bounding its idleness within one of its presence intervals. For conciseness, we denote this interval and its corresponding task as $[IT(T), ET(T))$ and T, respectively. If processor k is idle at any time in $[IT(T), ET(T))$, then some job of T is suspended at that time. Thus, the total suspension time of jobs of T in $[IT(T), ET(T))$, denoted $I(T)$, upper-bounds the idle time on processor k in $[IT(T), ET(T))$.

Task T may have multiple jobs that are enabled within its presence interval. Such a job is said to *fully execute* in the presence interval if it starts its first phase (be it execution or suspension) within the presence interval and also completes all of its execution phases in that interval (note that it may not complete all of its suspension phases). A job is said to *partially execute* in the presence interval if it starts its first phase (be it execution or suspension) before the presence interval or completes its last execution phase after the presence interval. Note that at most two jobs of T could partially execute in its presence interval (namely, the first and last jobs to be enabled in that interval). We now prove that $I(T)$, and hence the idleness within $[IT(T), ET(T))$ on processor k, is at most $\xi_{max}^H \cdot (ET(T) - IT(T)) + S_{max}^H + 2S_{max}^1$ (see Def. 13). Depending on the number of jobs of T that execute during T's presence interval, we have two cases.

Case 1. *T has at most H jobs that fully execute in its presence interval.* (Additionally, T may have at most two jobs that partially execute in its presence interval.) In this case, $I(T)$ is clearly at most $S_{max}^H + 2S_{max}^1$.

Case 2. *T has more than H jobs that fully execute in its presence interval.* (Again, T may have at most two jobs that partially execute in its presence interval.) In this case, the jobs of T that are enabled in its presence interval can be divided into n_T sets, where one set contains fewer than H fully executed jobs plus at most two partially-executed jobs and each of the remaining $n_T - 1$ sets contains exactly H fully-executed jobs. Let θ denote the union of the latter $n_T - 1$ job sets. Without loss of generality, we assume that θ contains jobs that are enabled consecutively. The total suspension time for the first job set defined above is clearly at most $S_{max}^H + 2S_{max}^1$. We complete this case by showing that the total suspension time for all jobs in θ is at most $\xi_{max}^H \cdot (ET(T) - IT(T))$.

To ease the analysis, let $IT'(T)$ be the start time of the first enabled job in θ, and $ET'(T) = min(ET(T), FT)$, where FT is the finish time of the last enabled job in θ. Then $ET'(T) - IT'(T) \leq ET(T) - IT(T)$. Also, $ET'(T) - IT'(T) = I(\theta) + \Delta(\theta) + E(\theta)$, where $I(\theta)$ is the total suspension time of all jobs in θ within $[IT'(T), ET'(T))$, $\Delta(\theta)$ is the total preemption time of all jobs in θ within $[IT'(T), ET'(T))$, and $E(\theta)$ is the total execution time of all jobs in θ within $[IT'(T), ET'(T))$. (Recall that, by Def. 11, a suspended task is not considered to be preempted.) Given that θ contains $(n_T - 1)$ H fully-executed jobs, $I(\theta) \leq (n_T - 1) \cdot S_T^H$. Moreover, given our assumption that each job executes for the corresponding task's worst-case execution time, $E(\theta) = (n_T - 1) \cdot H \cdot e(T)$, where $e(T)$ is the worst-case execution time of T. Thus,

$$
\begin{aligned}
I(\theta) &= \frac{I(\theta)}{ET'(T) - IT'(T)} \cdot (ET'(T) - IT'(T)) \\
&= \frac{I(\theta)}{I(\theta) + \Delta(\theta) + E(\theta)} \cdot (ET'(T) - IT'(T)) \\
&\leq \frac{I(\theta)}{I(\theta) + E(\theta)} \cdot (ET'(T) - IT'(T)) \\
&\quad \{ \text{because } I(\theta) \leq (n_T - 1) \cdot S_T^H \} \\
&\leq \frac{(n_T - 1) \cdot S_T^H}{(n_T - 1) \cdot S_T^H + E(\theta)} \cdot (ET'(T) - IT'(T)) \\
&\quad \{ \text{because } E(\theta) = (n_T - 1) \cdot H \cdot e(T) \} \\
&\leq \frac{(n_T - 1) \cdot S_T^H \cdot (ET'(T) - IT'(T))}{(n_T - 1) \cdot S_T^H + (n_T - 1) \cdot H \cdot e(T)} \\
&= \frac{S_T^H}{S_T^H + H \cdot e(T)} \cdot (ET'(T) - IT'(T)) \\
&\quad \{\text{by Def. 13}\} \\
&\leq \frac{S_{max}^H}{S_{max}^H + H \cdot e(T)} \cdot (ET'(T) - IT'(T)) \\
&\quad \{\text{by Def. 14}\} \\
&= \xi_{max}^H \cdot (ET'(T) - IT'(T)) \\
&\leq \xi_{max}^H \cdot (ET(T) - IT(T)).
\end{aligned}
$$

This concludes the proof of Case 2 of Claim 3.

Given that a task can be identified with only one presence interval and there are at most n tasks, the idleness within $[t_s, t_p)$ on m processor satisfies

$$
\begin{aligned}
I &\leq n \cdot (S_{max}^H + 2S_{max}^1) \\
&\quad + \sum_{T \in \lambda^{(1)} \cup \ldots \cup \lambda^{(k)}} \xi_{max}^H \cdot (ET(T) - IT(T)) \\
&= \xi_{max}^H \cdot m(t_p - t_s) + n \cdot (S_{max}^H + 2S_{max}^1).
\end{aligned}
$$

[1] As time increases from t_s to t_p, whenever a presence interval ends on processor k, a task exists that can be used to define the next presence interval on processor k. It can also be assumed, without loss of generality, that this task executes only on processor k during its presence interval.

Thus, $W' = m(t_p - t_s) - I \geq (1 - \xi^H_{max}) \cdot m(t_p - t_s) - n \cdot (S^H_{max} + 2S^1_{max})$. This completes the proof of Claim 3. □

We now complete the proof of Subcase 3.2 (and thereby Lemma 4). As shown in Fig. 4, $[t_d, t_s)$ and $[t_p, S(T_{l,j}))$ are busy for $\mathbf{d} \cup \mathbf{D}$. By Claim 2, the amount of work due to $\mathbf{d} \cup \mathbf{D}$ performed in $[S(T_{l,j}), F(T_{l,j}))$ is at least $m(F(T_{l,j}) - S(T_{l,j})) - (m-1)e_l - m \cdot s_l$. By Claim 3, the amount of work due to $\mathbf{d} \cup \mathbf{D}$ performed in $[t_s, t_p)$ is at least $(1 - \xi^H_{max}) \cdot m(t_p - t_s) - n \cdot (S^H_{max} - 2S^1_{max})$. By summing over all of these subintervals, we can lower-bound the amount of work due to $\mathbf{d} \cup \mathbf{D}$ performed in $[t_d, F(T_{l,j}))$, i.e., Z:

$$\begin{aligned} Z \quad \geq \quad & m(t_s - t_d) + (1 - \xi^H_{max}) \cdot m(t_p - t_s) \\ & - n \cdot (S^H_{max} + 2S^1_{max}) + m(S(T_{l,j}) - t_p) \\ & + m(F(T_{l,j}) - S(T_{l,j})) - (m-1)e_l - m \cdot s_l. \end{aligned} \tag{8}$$

By (6) and (8), we therefore have

$$\begin{aligned} & (1 - \xi^H_{max}) \cdot mx - (m-1)e_l - m \cdot s_l \\ & - n \cdot (S^H_{max} + 2S^1_{max}) \\ \geq \quad & m(t_s - t_d) + (1 - \xi^H_{max}) \cdot m(t_p - t_s) \\ & - n \cdot (S^H_{max} + 2S^1_{max}) + m(S(T_{l,j}) - t_p) \\ & + m(F(T_{l,j}) - S(T_{l,j})) - (m-1)e_l - m \cdot s_l, \end{aligned}$$

which gives,

$$F(T_{l,j}) - t_d \quad \leq \quad (1 - \xi^H_{max}) \cdot x + \xi^H_{max} \cdot (t_p - t_s).$$

According to Property (P) (applied to $T_{l,j}$'s predecessor), $t_p - t_s \leq t_p - t_d \leq x - p_l + e_l + s_l \leq x$. Therefore, $F(T_{l,j}) - t_d \leq (1 - \xi^H_{max}) \cdot x + \xi^H_{max} \cdot x < x + e_l + s_l$. □

3.3 Determining x

Setting the upper bound on $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S)$ in (5) to be at most the lower bound in Lemma 4 will ensure that the tardiness of $T_{l,j}$ is at most $x + e_l + s_l$. The resulting inequality can be used to determine a value for x. By (5) and Lemma 4, this inequality is $(U^s_{sum} + U^c_L) \cdot x + E^s_{sum} + E^c_L + u^s_{max} \cdot S^s_{sum} + E_{sum} \leq (1 - \xi^H_{max}) \cdot mx - (m-1)e_l - m \cdot s_l - n \cdot (S^H_{max} + 2S^1_{max})$.

Let $V = E^s_{sum} + E^c_L + u^s_{max} \cdot S^s_{sum} + E_{sum} + (m-1)e_l + m \cdot s_l + n \cdot (S^H_{max} + 2S^1_{max})$. Solving for x, we have

$$x \geq \frac{V}{(1 - \xi^H_{max}) \cdot m - U^s_{sum} - U^c_L}. \tag{9}$$

x is well-defined provided $U^s_{sum} + U^c_L < (1 - \xi^H_{max}) \cdot m$. If this condition holds and x equals the right-hand side of (9), then the tardiness of $T_{l,j}$ will not exceed $x + e_l + s_l$. A value for x that is independent of the parameters of T_l can be obtained by replacing $(m-1)e_l + m \cdot s_l$ with $max_l((m-1)e_l + m \cdot s_l)$ in V.

Theorem 1. *With x as defined in (9), the tardiness of any task T_l scheduled under GSA is at most $x + e_l + s_l$, provided $U^s_{sum} + U^c_L < (1 - \xi^H_{max}) \cdot m$.*

For GFIFO and GEDF, the bound in Theorem 1 can be improved.

Corollary 1. *For GFIFO, Theorem 1 holds with V replaced by $V - E_{sum} + \sum_{p_i > p_l} e_i$ in the numerator of (9).*

Proof. Under GFIFO, D_{CI} consists of carry-in jobs that are released before $r_{l,j}$ and have deadlines later than t_d, which implies that these jobs have periods greater than p_l. Thus, the upper bound in (5) can be refined to obtain $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S) \leq (U^s_{sum} + U^c_L) \cdot x + E^s_{sum} + E^c_L + u^s_{max} \cdot S^s_{sum} + \sum_{p_i > p_l} e_i$. Using this upper bound to solve for x, the corollary follows. □

Corollary 2. *For GEDF, Theorem 1 holds with V replaced by $V - E_{sum}$ in the numerator of (9).*

Proof. Under GEDF, the demand placed by jobs in $\mathbf{D}$ after t_d is zero because $\mathbf{D} = \emptyset$. Thus, under GEDF, $LAG(\mathbf{d}, t_d, S) + B(\mathbf{D}, t_d, S) \leq (U^s_{sum} + U^c_L) \cdot x + E^s_{sum} + E^c_L + u^s_{max} \cdot S^s_{sum}$. Using this upper bound to solve for x, the corollary follows. □

3.4 A Counterexample

Previous research has shown that every sporadic task system for which $U_{sum} \leq m$ without self-suspensions has bounded tardiness under GEDF and GFIFO [2, 7]. We now show that it is possible for a task system containing self-suspending tasks to have unbounded tardiness under GEDF or GFIFO if the utilization constraint in Theorem 1 is violated.

Consider a two-processor task set τ that consists of three self-suspending tasks: T_1 = ((3(exec.), 7(susp.)), 10(period)), T_2 = ((1(exec.), 8(susp.), 1(exec.)), 10(period)), and T_3 = ((1(exec.), 8(susp.), 1(exec.)), 10(period)). For this system, $\xi^H_{max} = 0.8$ (assuming $H = 1$) and $U^s_{sum} + U^c_L = 0.7$. Thus, $(1 - \xi^H_{max}) \cdot m = 0.4 < U^s_{sum} + U^c_L$, which violates the condition stated in Theorem 1. Fig. 6 shows the tardiness of each task in this system under GFIFO/GEDF by job index assuming each job is released as early as possible. We have verified analytically that the tardiness growth rate seen in Fig. 6 continues indefinitely.

3.5 Experimental Evaluation

In this section, we describe experiments conducted using randomly-generated task sets to evaluate the applicability of the tardiness bound in Theorem 1. Our goal is to examine how restrictive the theorem's utilization cap is, and to compare it with another commonly-used approach, which we call *SuspToComp*, wherein all suspension phases are treated

Figure 6: Tardiness growth rates in counterexample.

per-task utilization \ suspension length		short suspensions $\xi_{max} = 0.05$	moderate suspensions $\xi_{max} = 0.2$	long suspensions $\xi_{max} = 0.5$
light	min:	2.6 *µs*	12 *µs*	50 *µs*
	avg:	197 *µs*	938 *µs*	3.75 *ms*
	max:	526 *µs*	2.5 *ms*	10 *ms*
medium	min:	263 *µs*	1.25 *ms*	5 *ms*
	avg:	789 *µs*	3.75 *ms*	15 *ms*
	max:	1.6 *ms*	7.5 *ms*	30 *ms*
heavy	min:	789 *µs*	3.75 *ms*	15 *ms*
	avg:	2.2 *ms*	10.3 *ms*	41.25 *ms*
	max:	4.2 *ms*	20 *ms*	80 *ms*

Table 2: Per-job suspension-length ranges.

as computation phases. From [2, 7], tardiness is bounded under SuspToComp provided $U_{sum} \leq m$ and $U_L \leq m$, where U_L is the sum of the $\min(m-1, n)$ largest task utilizations. (Note that, under SuspToComp, treating suspensions as computation causes utilizations to be higher.)

In our experiments, task sets were generated as follows. Task periods were uniformly distributed over [50*ms*,100*ms*]. Per-task utilizations were distributed differently for each experiment using three uniform distributions: [0.001,0.1] (light), [0.1,0.3] (medium), and [0.3,0.8] (heavy). Task execution costs were calculated from periods and utilizations. We varied U^s_{sum} as follows: $U^s_{sum} = 0.1 \cdot U_{sum}$ (suspensions are relatively infrequent), $U^s_{sum} = 0.4 \cdot U_{sum}$ (suspensions are moderately frequent), and $U^s_{sum} = 0.7 \cdot U_{sum}$ (suspensions are frequent). Moreover, we varied ξ_{max} as follows: 0.05 (suspensions are short), 0.2 (suspensions are moderate), and 0.5 (suspensions are long). Table. 2 shows suspension-length ranges generated by these parameters. We also varied U_{sum} within $\{1, 2, ..., 8\}$. For each combination of (u_{max}, U^s_{sum}, U_{sum}), 1,000 task sets were generated for an eight-processor system. For each generated system, soft real-time schedulability (i.e., the ability to ensure bounded tardiness) was checked under SuspToComp and using the condition stated in Theorem 1. In doing so, system overheads were ignored (factoring overheads into our analysis is beyond the scope of this paper).

The schedulability results that were obtained are shown in Fig. 8 (the organization of which is explained in the figure's caption). Each curve plots the fraction of the generated

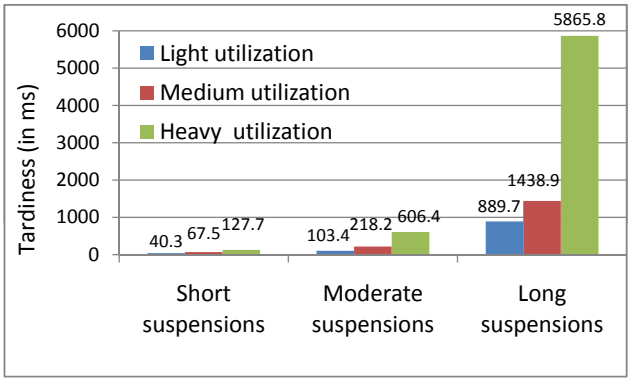

Figure 7: Average tardiness under LA, as computed via Theorem 1.

task sets the corresponding approach successfully scheduled, as a function of total utilization. As Fig. 8 shows, our approach proved to be superior, sometimes by a substantial margin, in all tested scenarios summarized in the first two rows of graphs. However, in many of the scenarios summarized in the third row of grows, SuspToComp proved to be superior. In these scenarios, task utilizations are high and suspensions are long or frequent. Our analysis is negatively impacted in such cases because U^c_L tends to be large when utilizations are high, and ξ_{max} tends to be large when suspensions are long. It is worth noting, however, that our approach allows certain tasks to be designated as computational tasks. Thus, the SuspToComp approach is really a special case of our approach. It would be interesting to investigate intermediate choices between the two extremes of modeling all versus no suspensions as computation.

In addition to schedulability, the magnitude of tardiness, as computed using the bound in Theorem 1, is of importance. Fig. 7 depicts the average of the computed bounds for each of the tested scenarios in our experimental framework for the case where $U_{sum} = m$ and $U^{sum}_s = 0.1 \cdot U_{sum}$ (that is, for each scenario in this case, an average of all bounds for all tasks in all schedulable task sets is plotted). As can be seen, tardiness is reasonable if task utilizations are low and suspensions are short. However, as either task utilizations or suspension lengths increase, tardiness increases, as an examination of the bound in Theorem 1 suggests should be the case. With large task utilizations and long suspension lengths, tardiness is quite high, perhaps unacceptably so, even though these systems are deemed to be schedulable.

4 Conclusion

We have derived a tardiness bound that can be applied to globally-scheduled sporadic task systems that include self-

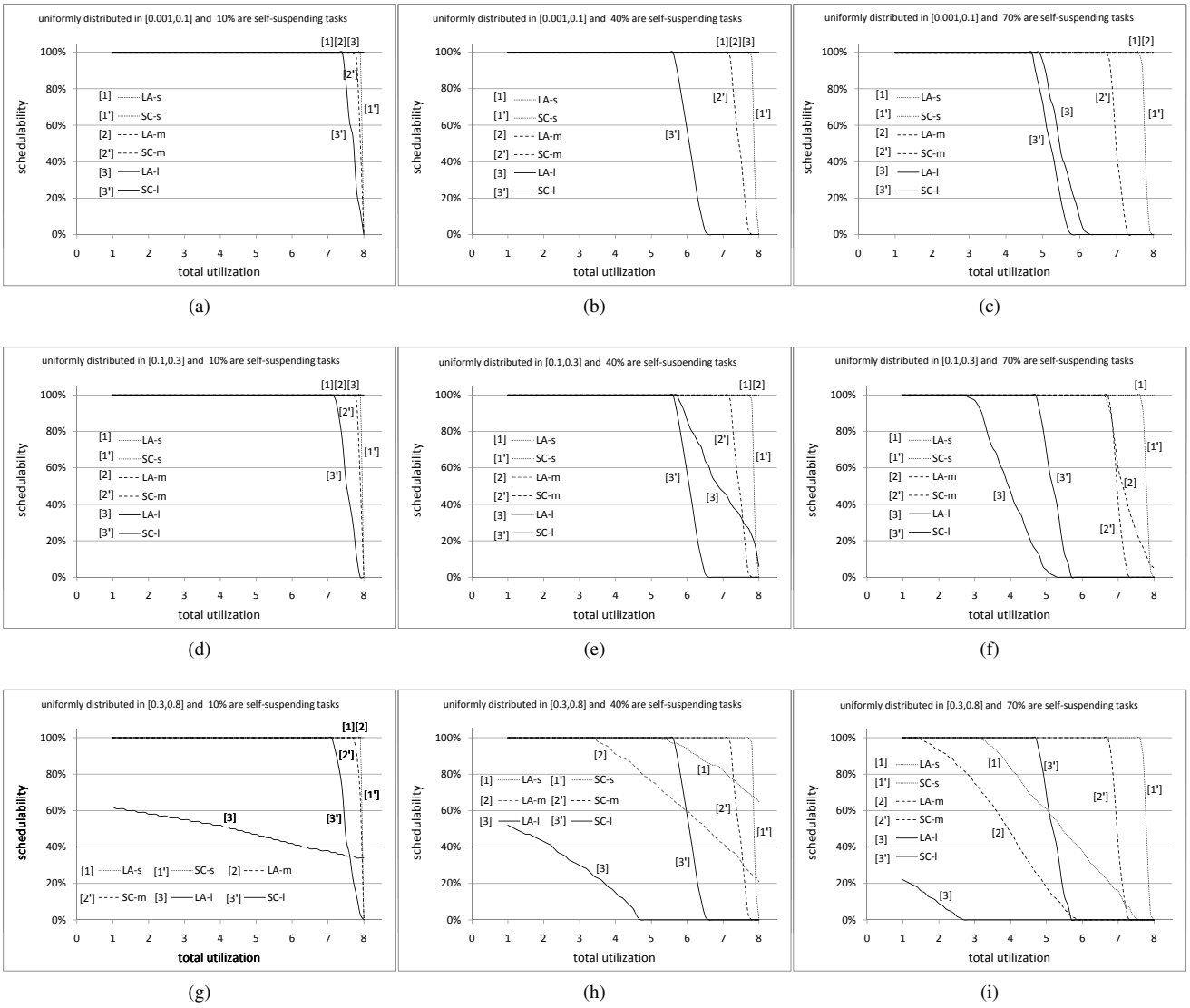

Figure 8: Soft real-time schedulability results. In the first (respectively, second and third) row of graphs, light (respectively, medium and heavy) per-task utilizations are assumed. In the first (respectively, second and third) column of graphs, relatively infrequent (respectively, moderately frequent and frequent) suspensions are assumed. Each graph gives three curves per tested approach for the cases of short, moderate, and long suspensions, respectively. The label "**LA-s(m/l)**" indicates the approach of this paper assuming short (moderate/long) suspensions. Similar "**SC**" labels are used for SuspToComp.

suspending tasks. This bound is applicable to a class of global algorithms that includes GEDF and GFIFO. The derived tardiness bound requires overall utilization to be constrained. We have shown via a counterexample that utilization constraints are fundamental. We also presented schedulability experiments that suggest that our constraint is quite liberal in many systems and is often less pessimistic than modeling suspensions as computation.

References

[1] U. Devi. An improved schedulability test for uniprocessor periodic task systems. In *Proc. of the 15th Euromicro Conf. on Real-Time Systems*, pp. 23-30, 2003.

[2] U. Devi and J. Anderson. Tardiness bounds under global EDF scheduling on a multiprocessor. In *Proc. of the 26th IEEE Real-Time Systems Symp.*, pp. 330-341, 2005.

[3] W. Kang, S. Son, J. Stankovic, and M. Amirijoo. I/O-aware deadline miss ratio management in real-time embed-

ded databases. In *Proc. of the 28th IEEE Real-Time Systems Symp.*, pp. 277-287, 2007.

[4] I. Kim, K. Choi, S. Park, D. Kim, and M. Hong. Real-time scheduling of tasks that contain the external blocking intervals. In *Proc. of the 2nd Int'l Workshop on Real-Time Computing Systems and Applications*, pp. 54-59, 1995.

[5] S. Lee and B. Moon. Design of flash-based DBMS: An in-page logging approach. In *Proc. of the 2007 ACM SIGMOD Conf. on Management of Data*, pp. 55-66, 2007.

[6] H. Leontyev and J. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. In *Proc. of the 28th IEEE Real-Time Systems Symp.*, pp. 413-422, 2007.

[7] H. Leontyev and J. Anderson. Tardiness bounds for FIFO scheduling on multiprocessors. In *Proc. of the 19th Euromicro Conf. on Real-Time Systems*, pp. 71-80, 2007.

[8] J. W. S. Liu. *Real-Time Systems*. Prentice Hall, 2000.

[9] J. C. Palencia and M. Gonzlez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *Proc. of the 19th IEEE Real-Time Systems Symp.*, pp. 26-37, 1998.

[10] J. C. Palencia and M. Gonzlez Harbour. Response time analysis of EDF distributed real-time systems. In *J. Embedded Comput.*, Vol.1, pp. 225-237, 2005.

[11] R. Rajkumar. Dealing with Suspending Periodic Tasks. IBM Thomas J. Watson Research Center, 1991.

[12] F. Ridouard and P. Richard. Worst-case analysis of feasibility tests for self-suspending tasks. In *Proc. of the 14th Real-Time and Network Systems*, pp. 15-24, 2006.

[13] F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Proc. of the 25th IEEE Real-Time Systems Symp.*, pp. 47-56, 2004.

5 Appendix

In this appendix, we prove Claim 4 and Lemmas 1 and 2. In proving Claim 4, we lift the restriction, given earlier in Sec. 2, that each job executes for the corresponding task's worst-case execution time. To state this claim, some additional terminology is required.

We say that a sporadic task system τ is *concrete* if the release time (and hence deadline) and actual execution cost and suspension time of every job of each task is fixed. Two concrete task systems are *compatible* if they have the same jobs with the same release times (they can have different actual execution and suspension times). A concrete task system τ is *maximal* if the actual execution time of any job equals the corresponding task's worst-case execution time.

Claim 4. *For any concrete task system τ, there exists a compatible maximal concrete task system τ' such that, for any job $T_{i,k}$, its response time in the GSA schedule for τ' is at least its response time in the GSA schedule for τ.*

Proof. The existence of the desired maximal concrete system is demonstrated via a construction method in which computation phases are ranked as follows: **(i)** if $T_{i,k} \prec T_{x,y}$, then all computation phases of $T_{i,k}$ are ranked before all computation phases of $T_{x,y}$; **(ii)** earlier computation phases of $T_{i,k}$ are ranked before later computation phases of $T_{i,k}$. Let ϵ be a positive value that is small enough so that within any interval $[t, t+\epsilon)$, each processor schedules exactly one job or no job, and the actual and worst-case execution cost of any computation phase is a multiple of ϵ.

Consider a computation phase C of a job $T_{i,k}$ that is not maximal. We show that the length of C can be increased by ϵ by adding to the end of C a piece of computation ρ of length ϵ. In so doing, it may be necessary to reduce the length of a lower-ranked computation phase by ϵ and to reduce the length of a subsequent suspension phase (if any) of $T_{i,k}$. By inducting over all computation phases in rank order, and by iteratively increasing any non-maximal execution time by ϵ, we can obtain a compatible concrete task system that is maximal. The construction method will ensure that no job's response time is reduced.

Let $[t, t+\epsilon)$ denote the time interval where ρ should be added to the schedule (according to GSA). If, before adding ρ, task T_i is scheduled within $[t, t+\epsilon)$, then the computation phase of T_i executing at that time, call it C', is ranked lower than C. In this case, we can accommodate ρ by reducing C' in length by ϵ.[2] If C and C' are separated by a suspension phase, then the length of that suspension phase must be defined to be zero.

In the rest of the proof, we consider the other possibility: before adding ρ, T_i is *not* scheduled within $[t, t+\epsilon)$ (and hence, it is not scheduled in $[t', t+\epsilon)$, where t' is the completion time of C). In this case, if there is an idle processor in $[t, t+\epsilon)$, then ρ can be scheduled there without modifying the length of any lower-ranked computation phase. On the other hand, if there is no idle processor, then, as ρ should be scheduled in $[t, t+\epsilon)$, there must be a computation phase ranked lower than C scheduled then. We can accommodate ρ and allow it to be scheduled in $[t, t+\epsilon)$ by reducing the length of that lower-ranked computation phase by ϵ. If C is followed by a suspension phase, then, once ρ has been added to the schedule, it may be necessary to reduce the length of that suspension phase. In particular, if, before adding ρ, $T_{i,k}$ was suspended in $[t, t+\epsilon)$, then the length of that suspension phase must be reduced so that it starts as $t+\epsilon$.

Note that the construction method used in this proof strongly exploits the fact that, in our task model, suspension phases are upper-bounded, and hence, can be reduced. □

[2] If C' is of length ϵ and is followed by a suspension phase, then we can avoid altering the length of that suspension phase by assuming that C' executes for zero time at time $t+\epsilon$. Note that C''s execution time will be increased in a subsequent induction step. A similar comment applies to the argument in the next paragraph.

Theorem 1 shows that, for any maximal concrete task system, the tardiness of any task T_l scheduled under GSA is at most $x + e_l + s_l$, with x as defined in (9). By Claim 4, the same is true for any non-maximal concrete task system.

Let $A(J, t_1, t_2, S)$ denote the total time allocated to all jobs in the job set J in $[t_1, t_2)$ in the schedule S.

Lemma 1. $LAG(\mathbf{d}, t_d, S) \leq LAG(\mathbf{d}, t_n, S) + \sum_{T_k \in D_{CI}} \delta_k(1 - u_k)$, where $t \in [0, t_d]$.

Proof. By (3), we have

$$\begin{aligned} LAG\left(\mathbf{d}, t_d, S\right) &\leq LAG(\mathbf{d}, t_n, S) + A\left(\mathbf{d}, t_n, t_d, PS\right) \\ &\quad - A\left(\mathbf{d}, t_n, t_d, S\right). \end{aligned} \tag{10}$$

We split $[t_n, t_d)$ into z non-overlapping intervals $[t_{p_i}, t_{q_i}), 1 \leq i \leq z$, such that $t_n = t_{p_1}, t_{q_{i-1}} = t_{p_i}$, and $t_{q_z} = t_d$. Each interval $[t_{p_i}, t_{q_i})$ is either busy or non-busy displacing for $\mathbf{d}$, by the selection of t_n. We assume that the intervals are defined so that for each non-busy displacing interval $[t_{p_i}, t_{q_i})$, if a task in D_{CI} executes in $[t_{p_i}, t_{q_i})$ then it executes continuously throughout $[t_{p_i}, t_{q_i})$; we let α_i denote the set of such tasks.

We now bound the difference between the work performed in the PS schedule and the GSA schedule S across each of these intervals $[t_{p_i}, t_{q_i})$. The sum of these bounds will give us a bound on the total allocation difference throughout $[t_n, t_d)$. Depending on the nature of the interval $[t_{p_i}, t_{q_i})$, two cases are possible.

Case 1. $[t_{p_i}, t_{q_i})$ is busy. Since in S all processors are occupied by jobs in $\mathbf{d}$, we have $A(\mathbf{d}, t_{p_i}, t_{q_i}, PS) - A(\mathbf{d}, t_{p_i}, t_{q_i}, S) \leq U_{sum}(t_{q_i}, t_{p_i}) - m(t_{q_i} - t_{p_i}) \leq 0$.

Case 2. $[t_{p_i}, t_{q_i})$ is non-busy displacing. The cumulative utilization of all tasks $T_k \in \alpha_i$ is $\sum_{T_k \in \alpha_i} u_k$. The carry-in jobs of these tasks do not belong to $\mathbf{d}$, by the definition of $\mathbf{d}$. Therefore, the allocation of jobs in $\mathbf{d}$ during $[t_{p_i}, t_{q_i})$ in PS is $A(\mathbf{d}, t_{p_i}, t_{q_i}, PS) \leq (t_{q_i} - t_{p_i})(m - \sum_{T_k \in \alpha_i} u_k)$. All processors are occupied at every time instant in the interval $[t_{p_i}, t_{q_i})$, because it is displacing. Thus, $A(\mathbf{d}, t_{p_i}, t_{q_i}, S) = (t_{q_i} - t_{p_i})(m - |\alpha_i|)$. Therefore, the allocation difference for jobs in $\mathbf{d}$ throughout the interval is

$$\begin{aligned} & A(\mathbf{d}, t_{p_i}, t_{q_i}, PS) - A(\mathbf{d}, t_{p_i}, t_{q_i}, S) \\ \leq\ & (t_{q_i} - t_{p_i}) \left((m - \sum_{T_k \in \alpha_i} u_k) - (m - |\alpha_i|)\right) \\ =\ & (t_{q_i} - t_{p_i}) \sum_{T_k \in \alpha_i} (1 - u_k). \end{aligned} \tag{11}$$

For each task T_k in D_{CI}, the sum of the lengths of the intervals $[t_{p_i}, t_{q_i})$ in which the carry-in job of T_k executes continuously is at most δ_k. Thus, summing the allocation differences for all the intervals $[t_{p_i}, t_{q_i})$ given by (11), we have

$$\begin{aligned} & A(\mathbf{d}, t_n, t_d, PS) - A(\mathbf{d}, t_n, t_d, S) \\ \leq\ & \sum_{i=1}^{z} \sum_{T_k \in D_{CI}} (t_{q_i} - t_{p_i})(1 - u_k) \\ \leq\ & \sum_{T_k \in D_{CI}} \delta_k(1 - u_k). \end{aligned} \tag{12}$$

Setting this value into (10), we get $LAG(\mathbf{d}, t_d, S) \leq LAG(\mathbf{d}, t_n, S) + A(\mathbf{d}, t_n, t_d, PS) - A(\mathbf{d}, t_n, t_d, S) \leq LAG(\mathbf{d}, t_n, S) + \sum_{T_k \in D_{CI}} \delta_k(1 - u_k)$. □

Lemma 2. $lag(T_i, t, S) \leq u_i \cdot x + e_i + u_i \cdot s_i$ for any task T_i and $t \in [0, t_d]$.

Proof. Let $d_{i,k}$ be the deadline of the earliest pending job of T_i, $T_{i,k}$, in the schedule S at time t. If such a job does not exist, then $lag(T_i, t, S) = 0$, and the lemma holds trivially. Let γ_i be the amount of work $T_{i,k}$ performs before t.

By the selection of $T_{i,k}$, we have

$$\begin{aligned} lag(T_i, t, S) &= \sum_{h \geq k} lag(T_{i,h}, t, S) \\ &= \sum_{h \geq k} (A(T_{i,h}, 0, t, PS) - A(T_{i,h}, 0, t, S)). \end{aligned}$$

Given that no job executes before its release time, $A(T_{i,h}, 0, t, S) = A(T_{i,h}, r_{i,h}, t, S)$. Thus,

$$\begin{aligned} lag(T_i, t, S) &= A(T_{i,k}, r_{i,h}, t, PS) - A(T_{i,k}, r_{i,k}, t, S) \\ &\quad + \sum_{h > i} (A(T_{i,h}, r_{i,h}, t, PS) \\ &\quad - A(T_{i,h}, r_{i,h}, t, S)). \end{aligned} \tag{13}$$

By the definition of PS, $A(T_{i,k}, r_{i,h}, t, PS) \leq e_i$, and $\sum_{h>k} A(T_{i,h}, r_{i,h}, t, PS) \leq u_i \cdot \max(0, t - d_{i,k})$. By the selection of $T_{i,k}$, $A(T_{i,k}, r_{i,k}, t, S) = \gamma_i$, and $\sum_{h>k} A(T_{i,h}, r_{i,h}, t, S) = 0$. By setting these values into (13), we have

$$lag(T_i, t, S) \leq e_i - \gamma_i + u_i \cdot \max(0, t - d_{i,k}). \tag{14}$$

There are two cases to consider.

Case 1. $d_{i,k} \geq t$. In this case, (14) implies $lag(T_i, t, S) \leq e_i - \gamma_i$, which implies $lag(T_i, t, S) \leq u_i \cdot x + e_i + u_i \cdot s_i$.

Case 2. $d_{i,k} < t$. In this case, because $t \leq t_d$ and $d_{l,j} = t_d$, $T_{i,k}$ is not the job $T_{l,j}$. Thus, by Property (P), $T_{i,k}$ has tardiness at most $x + e_i + s_i$, so $t + e_i - \gamma_i \leq d_{i,k} + x + e_i + s_i$. Thus, $t - d_{i,k} \leq x + \gamma_i + s_i$. Setting this value into (14), we have $lag(T_i, t, S) \leq u_i \cdot x + e_i + u_i \cdot s_i$. □