Response Time Bounds for G-EDF Without Intra-Task Precedence Constraints

Jeremy P. Erickson and James H. Anderson
University of North Carolina at Chapel Hill

Abstract. Prior work has provided bounds on the deadline tardiness that a set of sporadic real-time tasks may incur when scheduled using the global earliest-deadline-first (G-EDF) scheduling algorithm. Under the sporadic task model, it is necessary that no individual task overutilize a single processor and that the set of all tasks does not overutilize the set of all processors. In this work we generalize the task model by allowing jobs within a single task to run concurrently. In doing so we remove the requirement that no task overutilize a single processor. We also provide tardiness bounds that are better than those available with the standard sporadic task model.

1 Introduction

Multicore processors have been shown to be useful for supporting traditional soft real-time (SRT) workloads when bounded deadline tardiness is acceptable [1–4]. In this paper we extend these works to a broader class of workloads in which jobs are independent of each other and can be executed in parallel, such as servers handling independent requests. For both types of SRT workloads, temporal correctness requires that *tardiness bounds* exist, i.e., for each task, there exists an upper bound on the amount of time between the deadline of any job of that task and its actual completion time. Prior work has shown that the global earliest-deadline-first (G-EDF) algorithm is a good candidate scheduler when bounded tardiness is desired, as its use allows all available processing capacity to be utilized.

In most previous analysis of G-EDF, successive jobs (i.e. invocations) of each task are required to execute in sequence. This constraint arises naturally when jobs correspond to separate invocations of the same code segment. However, in some settings, jobs are released as separate threads in response to interrupts, in which case, successive jobs of the same task may execute concurrently. In prior hard real-time analysis of G-EDF [5], the impact of such concurrently-executing jobs has been considered, but to our knowledge, no such analysis exists for SRT systems for which bounded deadline tardiness is acceptable. Such analysis is the focus of this paper.

The task model considered in this paper is based on the widely-studied sporadic model, but differs from the usual specification of that model in two

ways. First, as implied by the discussion above, successive jobs of the same task are allowed to execute in parallel. Second, *early release* behavior [6] is allowed: a job may have an *actual release time* (or, *a-release time*) that is earlier than its *scheduler release time* (or, *s-release time*) A job's deadline is defined based on its s-release time, and constructive s-releases of each task τ_i are constrained to be no closer than T_i time units apart, where T_i is the minimum separation parameter of τ_i. However, a job may begin execution as early as its a-release time. These changes to the traditional sporadic model allow us to support general event models, as the following example illustrates.

Example In high-frequency trading systems, short response times are critical to minimize risk [7]. Consider such a system that responds to data from the market about two stocks. One stock is highly critical and should receive new information every 2 ms (but due to network uncertainty may not be timed precisely.) It may take up to 3 ms to process and should be processed as quickly as possible, so its deadline is 3 ms. Observe that this stock overutilizes a single processor and could not be supported using the traditional sporadic task model. However, it can be supported using the methodology provided in this paper. A second stock is less critical, should receive new information every 4 ms, and can take up to 2 ms to process. One possible execution on two processors is depicted in Fig. 1. Observe that the a-release times sometimes do occur before the s-release times (because incoming packets can arrive early or late) and that some jobs do miss deadlines.

The main contribution of this paper is to show that tardiness under G-EDF is greatly lessened if jobs of the same task are not constrained to execute in sequence. We show this by deriving per-job response-time bounds, from which tardiness bounds can be deduced. After deriving such bounds, we compare them experimentally to prior bounds, which were derived assuming no intra-task parallelism. We begin in the next section by more fully describing our system model.

2 System Model

We consider a system τ of n arbitrary-deadline sporadic tasks $\{\tau_1, \tau_2, \ldots, \tau_n\}$ running on m processors, with each task τ_i characterized by a worst-case execution time C_i, a minimum separation time (between s-releases) T_i, and a relative deadline D_i. No job may run concurrently with itself, but distinct jobs within the same task may run concurrently. In addition, we define a task's *utilization* $U_i \stackrel{\text{def}}{=} \frac{C_i}{T_i}$, the *task system utilization* $U(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} U_i$, and $m^+ \stackrel{\text{def}}{=} \lceil U(\tau) \rceil$.

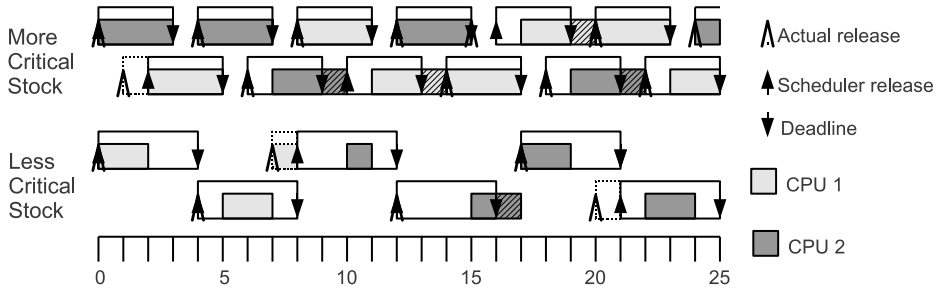


Fig. 1: Example high-frequency trading system

Under the traditional task model with implicit precedence constraints, providing bounded response time required that no τ_i had $U_i > 1$ and that $U(\tau) \leq m$ [3]. However, under the task model considered here, a job with $U_i > 1$ can have bounded response time if subsequent invocations run on separate processors, as depicted for τ_1 in Fig. 1. $U(\tau) \leq m$ remains necessary so that the entire system is not overutilized. In this work we demonstrate that $U(\tau) \leq m$ is also a sufficient condition for bounded response times and provide response-time bounds. Our bounds are relative to the s-release time of each job.

3 Response Time Characterization

Over an interval of any given length t, the total amount of work from jobs of τ_i (with both s-release times and deadlines inside the interval) is bounded. This bound, called the *demand bound function*, was first defined in [8]:

$$\text{DBF}(\tau_i, t) \stackrel{\text{def}}{=} C_i \cdot \max\left\{0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right\}. \tag{1}$$

(1) is defined by counting the number of possible jobs of τ_i having both s-release times and deadlines in an interval of length t, and multiplying that number by the worst-case execution time C_i. Because we still assume a sporadic s-release pattern for jobs, and deadlines are based solely on s-release times, allowing multiple jobs within a task to execute at the same time does not invalidate (1). An early release of a job can only reduce the demand as compared to that predicted in (1).

Lem. 1 of [3] used (1) to demonstrate that for all τ_i and $t \geq 0$,

$$\text{DBF}(\tau_i, t) \leq U_i t + S_i, \tag{2}$$

where $S_i \stackrel{\text{def}}{=} C_i \cdot \max\{0, 1 - C_i/D_i\}$. Essentially, S_i accounts for the extra demand that can be created by a job with a short deadline.

For an n-task system τ, we wish to define a vector of non-negative real numbers $\langle x_1, x_2, \ldots x_n \rangle$ such that the response time of each task τ_i, $1 \leq i \leq n$, is at most $x_i + C_i$ when τ is scheduled using G-EDF on m unit-speed processors. Each x_i value depends upon the other x_i values. Therefore, we initially define the vectors using an implicit criterion, and as in [2,3] we define the notion of a *compliant vector* as one that meets this criterion.

Definition 1. *For each task τ_i, non-negative integer $p < m^+ - 1$, and non-negative real number x_i, let*

$$l(\tau_i, x_i, p) = \min\{C_i, \max\{0, x_i + C_i - pT_i\}\}. \tag{3}$$

For any $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_n \rangle$, an ordered list of n non-negative real numbers, let

$$L(\boldsymbol{x}) \stackrel{def}{=} \sum_{m^+ - 1 \text{ largest}} l(\tau_i, x_i, p), \tag{4}$$

$$S(\tau) \stackrel{def}{=} \sum_{\tau_i \in \tau} S_i. \tag{5}$$

*We define $\boldsymbol{x}$ as a **compliant vector** if and only if*

$$\frac{L(\boldsymbol{x}) + S(\tau) + U(\tau)D_i - C_i}{m} \leq x_i \tag{6}$$

is satisfied for all i, $1 \leq i \leq n$.

Our definition differs from that of [3] both in the definition of $L(\boldsymbol{x})$ and in the additional $U(\tau)D_i$ term.

We now derive a response-time bound by considering a compliant vector $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_n \rangle$ and an arbitrary collection I' of jobs generated by τ. We order jobs by deadline with ties broken arbitrarily (as per the standard G-EDF algorithm). We analyze the response time of an arbitrary job J_k with s-release time r_k and deadline d_k, assuming that each job of each τ_i ordered prior to J_k completes within $(C_i + x_i)$ units of its s-release time. We denote as I the set of all jobs ordered at or before J_k, which (by the definition of G-EDF) contains all jobs that affect the scheduling of J_k. We also denote $I_c \stackrel{\text{def}}{=} I \setminus \{J_k\}$ (i.e., the work *competing* with J_k).

Without loss of generality, we assume that the earliest s-release time for any job in I is 0. We denote as $W_i(t)$ the remaining execution for jobs in I of task τ_i at time t, and let $W(t) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} W_i(t)$.

Lemma 1. *If $\boldsymbol{x}$ is a compliant vector and for all $i, 1 \leq i \leq N$, the response time of each job of τ_i in I_c is at most $x_i + C_i$, then*

$$W(t) \leq U(\tau)(d_k - t) + L(\boldsymbol{x}) + S(\tau). \tag{7}$$

Proof. We will define an interval as *busy* if at least m^+ processors are executing work throughout the interval, and *nonbusy* if fewer than m^+ processors are executing work. We will consider a set of time instants $\{t_0, t_1, \ldots t_k\}$, $t_0 = 0$, $t_k = r_k$, such that each $[t_i, t_{i+1})$ is either all busy or all nonbusy. We will prove the lemma by induction.

Base Case *($t_0 = 0$)* All jobs with both s-release times and deadlines within $[0, d_k]$ contribute to $W(0)$. By (2) $W_i(0) \leq U_i d_k + S_i$, and therefore, summing over all $W_i(0)$ values, $W(0) \leq U(\tau)d_k + S(\tau) \leq U(\tau)d_k + S(\tau) + L(\boldsymbol{x})$.

Induction Step Suppose the lemma is true for t_i. We will consider two subcases, based on whether $[t_i, t_{i+1})$ is busy or nonbusy.

Case A Suppose $[t_i, t_{i+1})$ is busy. Then,

$$
\begin{aligned}
& W(t_{i+1}) \\
\leq\ & \{\text{Since at least } m^+(t_{i+1} - t_i) \text{ work is completed}\} \\
& W(t_i) - m^+(t_{i+1} - t_i) \\
\leq\ & \{\text{By the inductive assumption}\} \\
& U(\tau)(d_k - t_i) + S(\tau) + L(\boldsymbol{x}) - m^+(t_{i+1} - t_i) \\
\leq\ & \{\text{Since } U(\tau) \leq m^+\} \\
& U(\tau)(d_k - t_i) + S(\tau) + L(\boldsymbol{x}) - U(\tau)(t_{i+1} - t_i) \\
=\ & \{\text{Simplifying}\} \\
& U(\tau)(d_k - t_{i+1}) + S(\tau) + L(\boldsymbol{x}),
\end{aligned}
$$

so the lemma is true for t_{i+1} as well.

Case B Suppose $[t_i, t_{i+1})$ is nonbusy. We will say that a job J is "executing at time instant t_{i+1}^-" if there is an ϵ greater than 0 such that J is executing over the entire interval $[t_{i+1} - \epsilon, t_{i+1})$. In [3], the presence of an idle CPU implied that at most $m^+ - 1$ *tasks* have work available for execution at time instant t_{i+1}^-, whereas here the same condition implies that at most $m^+ - 1$ *jobs* are available for execution. In [3] it was necessary to account for released jobs that were not running due to a precedence constraint, despite the presence of an idle CPU. In order to do so, assuming that $U_i \leq 1$ for each τ_i was necessary. Here we do not need to account for such a case, but do need to account for the fact that several jobs running in a non-busy interval could be from the same task. The assumption that $U_i \leq 1$ is no longer necessary.

We now consider two cases for jobs that may contribute to $W(t_{i+1})$: jobs that are executing at time instant t_{i+1}^- (Case B.1) and jobs that have s-release time at or after t_{i+1} (Case B.2).

B.1 In total, there may be at most $m^+ - 1$ jobs executing at time instant t_{i+1}^-. We ignore early-released jobs that have s-releases at or after t_{i+1}^-, as these are accounted for in Case B.2. We consider the jobs of each task τ_j that has jobs executing at time instant t_{i+1}^-. We will use p to index each executing job relative to the job with the most recent s-release within τ_j: $p = 0$ indicates the job with the most recent s-release, $p = 1$ the next most recent s-release, etc. By the assumption of the lemma, if $p > 0$ for job $J \in \tau_j$, then J must complete by $x_j + C_j$ units after its s-release time, and must be have a s-release time before $t_{i+1} - pT_j$. Therefore, J must complete by time $t_{i+1} + x_j + C_j - pT_j$, and its contribution to $W_j(t_{i+1})$ is at most $\min\{C_j, \max\{0, x_j + C_j - pT_j\}\} \stackrel{\text{By (3)}}{=} l(\tau_j, x_j, p)$.

When $p = 0$ for $J \in \tau_j$, $x_j + C_j - pT_j \geq C_j$. Therefore $l(\tau_j, x_j, p) = C_j$ by (3), so J's contribution to $W_j(t_{i+1})$ is also at most $l(\tau_j, x_j, p)$.

B.2 We now consider jobs with s-release time at or after t_{i+1}. By (2), each task τ_j contributes at most $U_j(d_k - t_{i+1}) + S_j$ units of work over $[t_{i+1}, d_k)$. Cumulatively, all tasks contribute at most $U(\tau)(d_k - t_{i+1}) + S(\tau)$ units of work over $[t_{i+1}, d_k)$.

Total $W(t_{i+1})$ contains at most $m^+ - 1$ jobs from Case B.1, in addition to all jobs from Case $B.2$, so $W(t_{i+1}) \leq U(\tau)(d_k - t_{i+1}) + S(\tau) + L(\boldsymbol{x})$.

Thus the lemma is true for t_{i+1}.

We now use the previous lemma to bound the response time of a job under the same assumptions.

Lemma 2. *If $\boldsymbol{x}$ is a compliant vector and for all $i, 1 \leq i \leq N$, the response time of each job of τ_i in I_c is at most $x_i + C_i$, then the response time of J_k is at most $x_k + C_k$.*

Proof. Recall that r_k is the s-release time of J_k, and d_k is its deadline. By Lem. 1,

$$W(r_k) \leq U(\tau)(d_k - r_k) + S(\tau) + L(\boldsymbol{x}). \tag{8}$$

After r_k, J_k is continuously running until it is finished, except when all other CPUs are occupied by jobs from I_c. Recall that, by definition, $W(r_k)$ is the total remaining work after time r_k for jobs in I. We define $W_c(r_k)$ as the total amount of remaining work after time r_k for jobs in I_c. Because the upper bound in (8) assumes that all jobs (including J_k) run for their full worst-case execution times, (8) implies

$$W_c(r_k) \leq U(\tau)(d_k - r_k) + S(\tau) + L(\boldsymbol{x}) - C_k. \tag{9}$$

The total amount of time after r_k during which m CPUs are busy with work from I_c can be at most

$$\begin{aligned}\frac{W_c(r_k)}{m} &\leq \{\text{By (9)}\} \\ &\quad \frac{L(\boldsymbol{x}) + S(\tau) + U(\tau)D_k - C_k}{m} \\ &\leq \{\text{By (6)}\} \\ &\quad x_k.\end{aligned}$$

Thus, J_k is prevented from executing after its s-release time for at most x_k time units, so its response time is at most $x_k + C_k$.

This lemma leads directly to the main result of this section:

Theorem 1. *If $\boldsymbol{x}$ is a compliant vector then $\forall i, 1 \leq i \leq N$, each job of τ_i completes within $x_i + C_i$ units of its s-release time.*

Proof. By inducting over the jobs of I' using Lem. 2.

4 The Minimum Compliant Vector

Thm. 1 uses compliant vectors to express response-time bounds. Our objective is to compute response-time bounds that are as small as possible. We show that for any arbitrary-deadline sporadic task system τ without implicit precedence constraints there exists a unique *minimum* compliant vector. This proof closely follows a similar one provided in [3], and some lemmas have nearly identical proofs. For space reasons, proofs of such lemmas are ommitted and included in an Appendix.[1] The Appendix also provides an algorithm for computing the minimum compliant vector in polynomial time.

We first characterize the behavior of $L(\boldsymbol{x})$. We consider two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ that differ by a constant for some of their values, and are the same elsewhere. For example, $\boldsymbol{x} = \langle 1, 2, 3\rangle$ and $\boldsymbol{y} = \langle 2, 2, 4\rangle$ differ by exactly 1 in two places (the first and third) and are the same in the second; Lem. 3 would apply to $\boldsymbol{x}$ and $\boldsymbol{y}$ with $k = 2$. Lem. 3 is proved in the Appendix.

Lemma 3. *Suppose length-n vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ differ at exactly k values, and for these values $y_i = x_i + \delta$, where δ is a positive constant. Denote $w = \min\{k, m^+ - 1\}$.*

Then, the following inequality holds:

$$L(\boldsymbol{x}) \leq L(\boldsymbol{y}) \leq L(\boldsymbol{x}) + \delta \cdot w. \tag{10}$$

[1] Available from http://cs.unc.edu/~anderson/papers.html

We say that length-n $\boldsymbol{x}$ is *strictly smaller* than length-n $\boldsymbol{y}$ if for all $i, x_i \leq y_i$ and there exists a j such that $x_j < y_j$. Clearly $\boldsymbol{y}$ cannot be considered "minimum" if there exists such an $\boldsymbol{x}$. We next use Lem. 3 (proved in the Appendix) to characterize the minimum compliant vector.

Lemma 4. *If $\boldsymbol{y}$ is compliant and there is a j such that $y_j > (L(\boldsymbol{y}) + S(\tau) + U(\tau)D_i - C_i)/m$, then there exists a strictly smaller vector $\boldsymbol{x}$ that is also compliant.*

Lem. 4 demonstrates that each inequality in (6) should actually be an equality, or the vector cannot be the minimum. A minimum compliant vector must therefore be of the form

$$x_i = \frac{L(\boldsymbol{x}) + S(\tau) + U(\tau)D_i - C_i}{m} \ \forall i. \tag{11}$$

Because $L(\boldsymbol{x})$ does not depend on i, there must exist a real number

$$s = \frac{L(\boldsymbol{x})}{m} \tag{12}$$

such that

$$x_i = s + \frac{S(\tau) + U(\tau)D_i - C_i}{m} \ \forall i. \tag{13}$$

We define some functions:

$$\boldsymbol{v}(s) \stackrel{\text{def}}{=} \boldsymbol{x} \text{ such that (13) holds} \tag{14}$$

$$L(s) \stackrel{\text{def}}{=} L(\boldsymbol{v}(s)) \tag{15}$$

$$M(s) \stackrel{\text{def}}{=} L(s) - ms. \tag{16}$$

By (13), any minimum compliant vector must be $\boldsymbol{v}(s)$ for some s. Furthermore, $L(s)$ must equal ms, by (12). Therefore, $M(s) = 0$ if and only if $\boldsymbol{v}(s)$ is a compliant vector in the form of (11), and thus the minimum compliant vector. We are now ready to prove this section's main result:

Theorem 2. *For any given task set τ, there exists a unique minimum compliant vector.*

Proof. We wish to demonstrate that exactly one real s exists such that $M(s) = 0$. We will use the Intermediate Value Theorem from calculus.

A necessary precondition for the Intermediate Value Theorem is that $M(s)$ is a continuous function. In the Appendix we provide the following lemma, which leads to the desired result as a corollary.

Lemma 21 *$L(s)$ is continuous over $\mathbb{R}$*

Let $C_{\max}$ denote the largest C_i value in τ. We now show that $M(0) > 0$ and $M(C_{\max}) < 0$, completing the preconditions for the Intermediate Value Theorem.

Lemma 22 *$M(0) > 0$*

Proof. Let $1 \leq i \leq N$ be arbitrary. Then:

$$
\begin{aligned}
& M(0) \\
= & \{\text{By (16) with } s = 0\} \\
& L(0) \\
= & \{\text{By (15) and (4)}\} \\
& \sum_{m^+ - 1 \text{ largest}} l(\tau_i, v_i(0), p) \\
\geq & \{\text{Since, by (3), } l(\tau_i, v_i(0), p) \text{ can't be negative}\} \\
& l(\tau_i, v_i(0), 0) \\
= & \{\text{By (3) and (14), with } s = 0 \text{ and } p = 0\} \\
& \min\left\{C_i, \max\left\{0, \frac{S(\tau) + U(\tau)D_i - C_i}{m} + C_i\right\}\right\} \\
= & \{\text{Simplifying}\} \\
= & \min\left\{C_i, \max\left\{0, \frac{S(\tau) + U(\tau)D_i + (m-1)C_i}{m}\right\}\right\} \\
> & 0.
\end{aligned}
$$

Lemma 23 *$M(C_{\max}) < 0$.*

Proof. By (3), $l(\tau_i, x_i, p) \leq C_i$ for any i and p. Therefore, for any i and p,

$$l(\tau_i, x_i, p) \leq C_{\max}. \tag{17}$$

Therefore,

$$
\begin{aligned}
& M(C_{\max}) \\
= & \{\text{By (16) with } s = C_{\max}\} \\
& L(C_{\max}) - mC_{\max} \\
\leq & \{\text{By (15), (4), and (17)}\} \\
& (m^+ - 1)C_{\max} - mC_{\max}
\end{aligned}
$$

$$
\begin{aligned}
&\leq \{\text{Since } m^+ \leq m\} \\
&\quad - C_{\max} \\
&< 0.
\end{aligned}
$$

Lemma 24 *There is an s in $(0, C_{\max})$ such that $M(s) = 0$.*

Proof. By Lem. 21, Lem. 22, Lem. 23, and the Intermediate Value Theorem.

We now verify that the s value of Lem. 24 is unique, using the following lemma, proved in the Appendix.

Lemma 25 *$s_1 \neq s_2$ implies $M(s_1) \neq M(s_2)$*

Lem. 25 demonstrates that $s_1 \neq s_2$ and $M(s_1) = 0$ imply $M(s_2) \neq 0$, so the value of s characterized in Lem. 24 is unique.

Here we have a substantial improvement compared to [3], where the upper bound was given as the sum of the $m^+ - 1$ largest values of C_i. This improvement leads to Thm. 3, which provides a response-time bound that can be quickly calculated.

Theorem 3. *The response time of any job of any task τ_i cannot exceed $C_{\max} + \frac{S(\tau)+U(\tau)D_i-C_i}{m} + C_i$.*

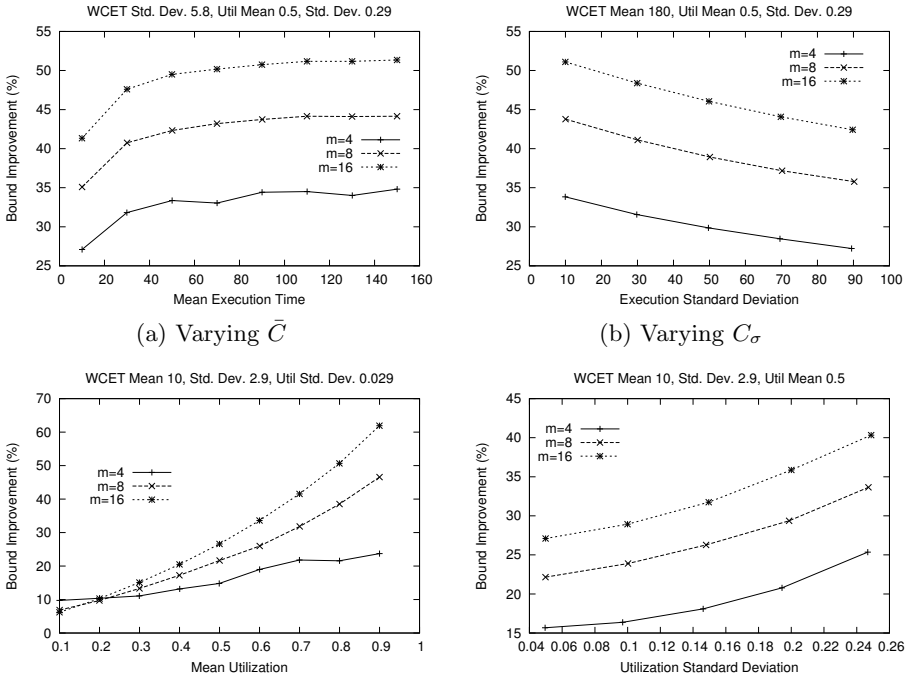
Proof. Follows from Lem. 24, (13), and Thm. 1.

5 Evaluation

This work allows smaller response-time bounds than are possible using prior work. In particular, these results are especially competitive for implicit-deadline sporadic task systems. By Thm. 3, combined with $U(\tau) \leq m$ (a necessary condition), and the fact that, for implicit-deadline systems, $S(\tau) = 0$, the response time of any job of any task τ_i must be upper-bounded by $C_{\max} + D_i + \frac{m-1}{m}C_i$. Therefore, the *tardiness* of any job of τ_i must be no greater than $C_{\max} + \frac{m-1}{m}C_i$.

In order to evaluate the improvement to the bounds we obtain by eliminating implicit precedence constraints, we compared our results to the best available analysis for implicit-deadline sporadic tasks, found in [2]. For the experiments in this paper, we compared the best results of our work to the best bounds attainable using [2].

Our experimental methodology is inspired by the tests in [2]. All experiments were done with processor counts of 4, 8, and 16. We used uniform



(a) Varying $\bar{C}$ (b) Varying C_σ

(c) Varying $\bar{U}$ (d) Varying U_σ

Fig. 2: Results of experiments

distributions for the task worst-case execution times and utilizations, and we determined the effects of varying each of four parameters: mean worst-case execution time ($\bar{C}$), standard deviation of worst-case execution time (C_σ), mean utilization ($\bar{U}$), and standard deviation of utilization (U_σ). For mean x and standard deviation σ, values were chosen uniformly over $(x - \sigma\sqrt{3}, x + \sigma\sqrt{3})$.

In each experiment, the processor count m and three of the four parameters above were fixed, and the remaining parameter was varied. For each value of the varied parameter, we generated 1000 task sets. For each individual task set, we generated tasks until a task was generated that would cause $U(\tau)$ to exceed m. For each task set we computed the mean tardiness bound with respect to [2], δ, and with respect to our work, δ'. For each set of 1000 task sets we computed $\bar{\delta}$ (the mean value of δ) and $\bar{\delta'}$ (the mean value of δ'). The *absolute improvement* for each set of sets is defined as $\bar{\delta} - \bar{\delta'}$, and the *relative improvement* for each set of sets is defined as $(\bar{\delta} - \bar{\delta'})/\bar{\delta}$.

Results are in Fig. 2. We see that the improvement to tardiness is quite substantial, particularly with large execution times, small variance in execution times, large utilizations, and large variance in utilizations. More significant improvement occurs with larger processor counts because the bounds

of [2] increase significantly with m, while our bounds are upper-bounded by $C_{\max} + \frac{m-1}{m}C_i$. This improvement is possible even when per-task utilization is restricted to be less than one to make our results comparable to prior work. We do not have results comparing our work to previous results when per-task utilization may exceed one, because prior work is not applicable in this case.

6 Conclusion

G-EDF scheduling has already proven useful for traditional SRT workloads in which jobs of the same task have implicit precedence constraints. Here we have demonstrated that G-EDF scheduling may be even more useful for SRT workloads in which jobs may be released as separate threads that can safely run concurrently. We have shown that doing so not only improves response times compared to prior work, but enables new workloads where a single task may overutilize a single processor.

For future work, allowing critical sections would be useful, so that tasks that write shared data but do not have precedence constraints could be handled. Supporting integrated workloads where some tasks have internal precedence constraints and some do not would also be interesting to consider. Furthermore, in past work on slack reclaiming, precedence constraints between jobs have prevented slack produced by a job from being reclaimed after its successor is released. Using the methods in this paper, we may be able to overcome this limitation.

References

1. Devi, U.C., Anderson, J.H.: Tardiness bounds under global EDF scheduling on a multiprocessor. The Journal of Real-Time Systems **38**(2) (2008) 133–189
2. Erickson, J.P., Devi, U.C., Baruah, S.K.: Improved tardiness bounds for global EDF. In: ECRTS. (2010) 14–23
3. Erickson, J.P., Guan, N., Baruah, S.K.: Tardiness bounds for global EDF with deadlines different from periods. In: OPODIS. (2010) 286–301
4. Leontyev, H., Anderson, J.H.: Generalized tardiness bounds for global multiprocessor scheduling. The Journal of Real-Time Systems **44**(1) (February 2010) 26–71
5. Baker, T., Baruah, S.K.: An analysis of global EDF schedulability for arbitrary-deadline sporadic task systems. The Journal of Real-Time Systems **43**(1) (2009) 3–24
6. Anderson, J.H., Srinivasan, A.: Early-release fair scheduling. In: ECRTS. (2000) 35–43
7. Durbin, M.: All About High-Frequency Trading. 1 edn. McGraw-Hill (2010)
8. Baruah, S.K., Mok, A.K., Rosier, L.E.: Preemptively scheduling hard-real-time sporadic tasks on one processor. In: RTSS. (1990) 182–190