Beating G-EDF at its Own Game: New Results on G-EDF-like Schedulers*

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

Abstract

The global earliest-deadline-first (G-EDF) scheduling algorithm is useful for both hard real-time and soft real-time systems. Under G-EDF, each job is prioritized based on a priority point that is a fixed time after its release, namely its deadline. By considering other such priority points, we define and examine a class of "G-EDF-like" scheduling algorithms and demonstrate that G-EDF is suboptimal in this class for hard real-time task systems. We also provide experimental evidence that soft real-time performance can be substantially improved by choosing priority points other than deadlines. We further provide a sufficient polynomial-time algorithm for determining priority points that will guarantee given response-time bounds, if possible.

1 Introduction

The earliest-deadline first (EDF) scheduling algorithm has been studied widely. Liu and Layland [1] demonstrated that, on uniprocessors, it is an *optimal* algorithm capable of scheduling any feasible set of jobs. Multiprocessor extensions have also been studied extensively, in particular in the form of the *global EDF (G-EDF)* scheduling algorithm, which on m processors always executes the jobs with the m soonest deadlines. Researchers have studied the use of G-EDF for both *hard real-time (HRT)* systems, where correctness requires meeting all deadlines [2], and *soft real-time (SRT)* systems, where missing deadlines by a bounded amount is acceptable [3, 4, 5].

Although G-EDF is known to be suboptimal on multiprocessors, it is attractive for several reasons. Optimal algorithms, such as the Pfair family of algorithms [6], cause tasks to experience frequent preemptions and migrations, and the resulting overheads can be prohibitive. In contrast, experimental research has demonstrated that the overheads caused by G-EDF are reasonable when it is used on a moderate number of processors [7]. Furthermore, unlike optimal algorithms, G-EDF is a *job-level static-priority (JLSP)* algorithm, and such algorithms are the target of most known work on real-time synchronization algorithms. Several promising non-optimal schedulers such as the earliest-deadline-until-zero-laxity (EDZL) algorithm [8, 9] are also not JLSP and thus cannot be used with these synchronization algorithms.

In [10], Leontyev and Anderson observed that scheduling priorities for most schedulers can be modeled by giving each job a *priority point (PP)* in time, with the scheduler always picking the job with the earliest PP (with appropriate tie-breaking). For example, fixed-priority task systems can be modeled by assigning all jobs of each task a single PP near the beginning of the schedule. G-EDF can be modeled by defining the absolute deadline of a job as its PP. In [11], response times are analyzed for many schedulers, including a class of *G-EDF-like (GEL)* schedulers, in which the PP of each job is defined as some per-task constant after the job's release. However, the response times provided were not intended to be tight. Furthermore, no comparison of differing GEL schedulers is provided.

In this paper, we demonstrate that G-EDF is *suboptimal* for HRT scheduling within the class of GEL schedulers, i.e., there exist task systems schedulable using some GEL scheduler, but not using G-EDF. For SRT scheduling, G-EDF is "optimal" in the sense that bounded tardiness can be ensured with no utilization loss. However, smaller tardiness can be achieved by using a different GEL scheduler. We demonstrate experimentally that a particular GEL scheduler, in which each job's PP is the earliest point where it may reach

*Work supported by Northrop Grumman Corporation, NSF grants CNS 0834270, CNS 0834132, and CNS 1016954; ARO grant W911NF-09-1-0535; AFOSR grant FA9550-09-1-0549; and AFRL grant FA8750-11-1-0033.

zero laxity, is typically superior to G-EDF for SRT scheduling, both in terms of analytically-derived tardiness bounds and actual runtime tardiness.

In the results described so far, PP definitions are assumed to be given. In the last part of the paper, we turn our attention to the problem of finding a good allocation of PP offsets to tasks. We provide a polynomial-time algorithm to determine whether it is possible to define such offsets to ensure particular response-time bounds using existing analysis. This algorithm is applicable to both HRT and SRT systems, as well as systems with a mix of HRT and SRT tasks.

The main conclusion to be drawn from our work is that HRT schedulability and SRT tardiness bounds and behavior can be improved while only slightly modifying the existing runtime implementation of G-EDF and maintaining its JLSP property. All prior implementation work on arbitrary-deadline G-EDF and all theoretical work assuming JLSP scheduling continue to apply when such modifications are made.

Because our work involves shortening the deadline used by the scheduler, it superficially resembles the work of Lee et al. [12], in which the deadlines of some tasks are shortened at design time to create "contention-free slots" that allow the priorities of some jobs to be lowered during runtime, increasing system schedulability. However, in their work, the actual deadlines by which jobs must complete are altered, which is not true of our work. Furthermore, their work requires modifying G-EDF in a manner that adds additional runtime overhead and removes the JLSP property, while our work does not.

In Sec. 2, we describe the task model under consideration and define basic terms. In Sec. 3, we demonstrate that G-EDF is suboptimal for HRT scheduling in the class of GEL schedulers and can be improved upon for SRT scheduling as well. In Sec. 4, we provide a polynomial-time algorithm for assigning PP offsets to tasks in order to ensure specific response-time bounds.

2 Task Model

We consider a system τ of n *arbitrary-deadline* sporadic tasks $\tau_i = (T_i, C_i, D_i)$ running on m processors, where T_i is the minimum separation time between subsequent releases of jobs of τ_i, C_i is the worst-case execution time of any job of τ_i, and D_i is the relative deadline of each job of τ_i. We use $U_i = C_i/T_i$ to denote the *utilization* of τ_i. We consider continuous time. Also, we assume that $n > m$. If this is not the case, then each task can be assigned its own processor, and no job of τ_i will have response time exceeding C_i.

If a job of τ_i has absolute deadline d and has executed for δ time units at time t, then its *laxity* is $(d - t) - (C_i - \delta)$. If such a job completes execution at time t, then its *tardiness* is $\max\{0, t - d\}$. For HRT scheduling, the tardiness of all jobs must be zero, whereas for SRT scheduling, a bound must exist on the tardiness of all jobs. Note that if a job reaches a *zero-laxity* point, then it must be continuously scheduled until its completion, or it may miss a deadline.

When analyzing GEL schedules, we use Y_i to denote the relative PP for task τ_i (e.g., for G-EDF, $Y_i = D_i$). In order to be able to apply the results in Sec. 3.2, we require that for all i, $Y_i \geq 0$. We let $\vec{Y} = \langle Y_1, Y_2, \ldots, Y_n \rangle$.

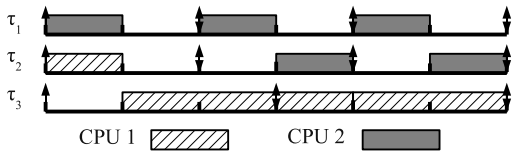
3 Inferiority of G-EDF

In this section, we demonstrate that G-EDF is inferior to other GEL schedulers. In Sec. 3.1, we provide an example of an implicit-deadline sporadic task system that is HRT-schedulable using a GEL algorithm, but not using G-EDF. In Sec 3.2, we provide experimental evidence that substantial improvement in SRT tardiness is possible compared to G-EDF within the GEL class.

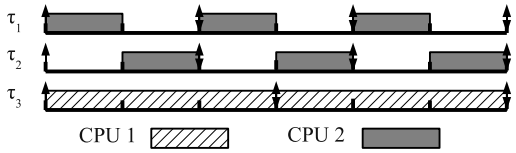
3.1 Hard Real-Time

Within the class of GEL schedulers, G-EDF is suboptimal for HRT systems even for *implicit-deadline* systems in which, for all i, $D_i = T_i$. Consider the system of $n = 3$ tasks, $\tau_1 = (2, 1, 2)$, $\tau_2 = (2, 1, 2)$, and $\tau_3 = (3, 3, 3)$, executing on $m = 2$ processors. Fig. 1 depicts the G-EDF schedule, as well as the GEL schedule with $Y_1 = 2$, $Y_2 = 2$, and $Y_3 = 0$, when job releases are periodic.

Observe that in the G-EDF schedule (Fig. 1(a)) the first job of τ_3 misses its deadline by one time unit. Therefore, τ is not schedulable using G-EDF. However, τ is schedulable using a GEL scheduler with $Y_1 = 2$, $Y_2 = 2$, and $Y_3 = 0$, as indicated by Fig. 1(b). In order to prove that τ is schedulable for all sporadic task release patterns, rather than just for the periodic system depicted in Fig. 1(b), we begin by observing that each job J of τ_3 is "tardy" from the perspective of the scheduler (i.e., it has already reached its PP) from the moment it is released. Therefore, J must be scheduled whenever it has been released but has not completed, unless some job of τ_1 or τ_2 is also past its PP. In the worst case, τ_3 will take up one processor, leaving one processor free for the execution of τ_1 and τ_2. Because $U_1 = 1/2$ and $U_2 = 1/2$, the Liu and Layland bound for uniprocessor EDF [1] ensures that τ_1 and τ_2



(a) G-EDF Schedule.



(b) GEL schedule with $Y_1 = 2$, $Y_2 = 2$, $Y_3 = 0$.

Figure 1: Example HRT task system when releases are periodic.

are schedulable on this remaining processor, and will therefore never execute past their PPs. If τ_3 releases sporadically or finishes before its full worst-case execution time, an extra processor may be available for scheduling τ_1 and τ_2, but the presence of this extra processor does not jeopardize the ability of τ_1 and τ_2 to meet their deadlines.

We have thus demonstrated that even for a simple implicit-deadline task system on two processors, whether periodic or sporadic, it is possible for the system to be schedulable using a GEL scheduler, but not using G-EDF. Therefore, the deadline is not the optimal place for the PP on a multiprocessor, even though [1] demonstrated that on a uniprocessor the deadline is the optimal place for the PP.

Intuitively, the reason G-EDF fails is that in this particular system τ_3 has a zero-laxity point at its release, but the standard G-EDF scheduler fails to account for it. While an EDZL scheduler would correctly schedule τ, it adds additional overhead compared to G-EDF, due to the need to track when a zero-laxity point occurs. EDZL also changes the priority of a job during its execution, so it does not have the JLSP property. By simply moving the PP for Y_3 back to reflect this zero-laxity point, we achieve the benefit of EDZL without incurring additional overhead or sacrificing JLSP.

3.2 Soft Real-Time

Unlike in the case of HRT, G-EDF can schedule any feasible set of SRT tasks. However, here we show that better GEL schedulers exist in the sense of creating smaller tardiness, both with respect to analytical bounds (with current analysis) and with respect to observed tardiness. Our SRT results extend prior work in [5], which analyzes the tardiness of arbitrary-deadline G-EDF. We summarize that work here, both so that we can explain the better analytical bounds possible for non-G-EDF GEL algorithms, and because we build on these results in Sec. 4. We begin by observing that all GEL algorithms can be analyzed as G-EDF with arbitrary deadlines by letting $D_i = Y_i$. The approach in [5] involves defining $\vec{x} = \langle x_1, x_2, \ldots, x_n \rangle$ such that no job of task τ_i has a tardiness greater than $x_i + C_i$. This implies that the response time of such a job is at most $x_i + C_i + D_i$, so in our analysis, where $D_i = Y_i$, the response time of such a job is at most

$$x_i + C_i + Y_i. \tag{1}$$

In order for a system to have bounded response times, both

$$\forall i, U_i \leq 1 \tag{2}$$

and

$$\sum_{\tau_i \in \tau} U_i \leq m \tag{3}$$

must hold. [5] demonstrates that these conditions are sufficient for bounded tardiness, and therefore for bounded response time.

Example. We will repeatedly consider a system θ of $n = 3$ tasks $\theta_1 = (10, 9, 10)$, $\theta_2 = (10, 9, 10)$, $\theta_3 = (100, 20, 90)$ running on $m = 2$ processors, to illustrate important ideas. Note that θ satisfies (2) and (3).

We now review how to derive $\vec{x}$ so that (1) holds.[1] To simplify the analysis, we assume that, for each τ_i, $Y_i \leq T_i$. Analysis for tasks for which $Y_i > T_i$ is similar to the $Y_i = T_i$ case.

In order to analyze the response time of a job J_k (from task τ_k) with absolute PP y_k, only jobs with PPs at or before y_k are considered, because the priority definition ensures that other jobs will not interfere with J_k. We inductively assume that no job of any τ_i with PP before y_k has response time exceeding $Y_i + x_i + C_i$, and then show that J_k has response time at most $Y_k + x_k + C_k$. We examine the last *idle interval* at or before y_k, here denoted as $[t_0, t)$, during which at least one CPU is idle. (We let $t = 0$ if there is no such interval.) A generic example schedule is depicted in 2(a).

We account for the work remaining for each task at time t. Each task τ_i will conform to one of three cases.

Case 1. In the simplest case, τ_i is not executing just before t, even though a processor is idle. Therefore, all jobs of τ_i that have not completed before t must

[1] Our intent here is to review the basic ideas from the analysis in [5]. For more details, please see [5].

be released after t. Examples of such tasks are highlighted in Fig. 2(b). We must therefore account for all jobs released at or after t and due at or before y_k. The worst-case release pattern for an example task is depicted in Fig. 2(c). For any task for which $Y_i = T_i$, the contributed work over $[t, y_k)$ can be upper-bounded by $\lfloor (y_k - t)/T_i \rfloor C_i \leq U_i(y_k - t)$, However, this is not true if $Y_i < T_i$, as depicted in Fig. 2(c), in which τ_i has three job releases (released after t and with PPs no later than y_k) even though $[t, y_k)$ is shorter than $3T_i$ time units. Therefore, for each task τ_j we define

$$S_j = \max\left\{0, C_j\left(1 - \frac{Y_j}{T_j}\right)\right\}, \tag{4}$$

so that the work released by τ_i is upper-bounded by

$$U_i(y_k - t) + S_i. \tag{5}$$

Case 2. It is also possible that τ_i runs at the end of $[t_0, t)$, but not throughout $[t_0, t)$. In this case, we define t_1 as the earliest time such that τ_i is running throughout $[t_1, t)$. This situation is depicted in Fig. 2(d). The analysis is the same as in Case 1, except that the releases of τ_i must be after t_1, rather than after t. However, because $t - t_1$ of this work is performed before t, the same upper bound,

$$U_i(y_k - t) + S_i, \tag{6}$$

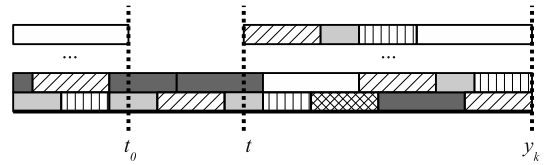
from Case 1, applies.

Case 3. Finally, τ_i may be running throughout an entire idle interval. In this case, we consider the job J running at time t_0. If J has its PP at or after t_0, then the situation depicted in Fig. 2(e) applies. In that case, the remaining work of τ_i from t_0 onward is upper-bounded by the remaining work of J (at most C_i), plus the work released after $t_0 + T_i - Y_i$. Because $t - t_0$ work must be completed in $[t_0, t)$, the remaining work after t is upper-bounded by $U_i(y_k - t) + C_i$.[2] Alternatively, if J has its PP before t_0, then the situation depicted in Fig. 2(f) occurs. Because $\forall j$, $Y_j \geq 0$, J cannot be preempted. Suppose J has $C_i - \delta$ units of execution left at t_0, for some non-negative δ. Then, it must complete at time $t_0 + C_i - \delta$, and it must have a PP no earlier than $x_i + C_i$ units prior to its completion (by the inductive assumption stated earlier). Because successor jobs cannot be released earlier than $T_i - Y_i$ units after that PP, they must be released no earlier than time $t_0 - \delta - x_i + T_i - Y_i$. Because $\delta \geq 0$, and $t - t_0$ work will be performed in $[t_0, t)$, the remaining work at time t will be at most

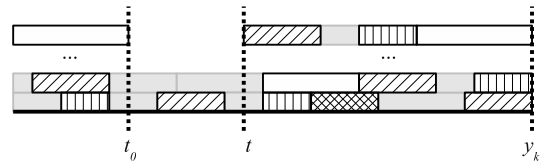
$$C_i + U_i(y_k - t + x_i).^3 \tag{7}$$

[2] In [5], the authors used a slightly less tight bound.

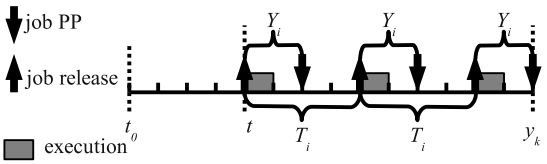
[3] As above, in [5] the authors used a slightly less tight bound.



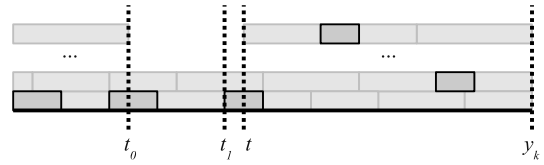
(a) Example schedule depicting t_0, t, y_k.



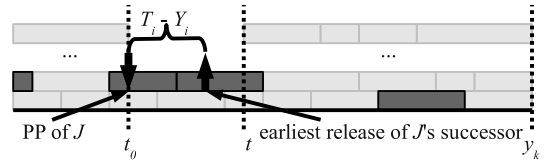
(b) Tasks not running at end of $[t_0, t)$.



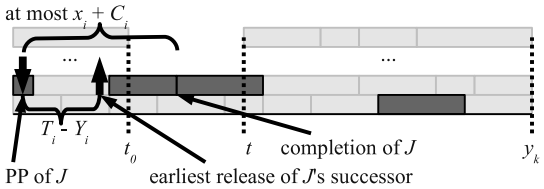
(c) Worst-case release pattern for $\tau_i = (4, 1, 2)$ when $Y_i = 2$.



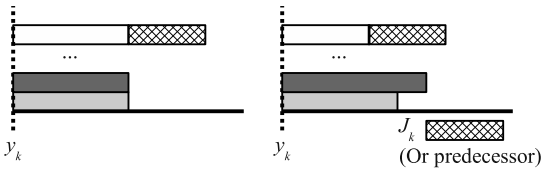
(d) Task running at end of, but not for entire, idle interval.



(e) Task running for entire idle interval, J not past PP at t_0.



(f) Task running for entire idle interval, J past PP at t_0.



(g) Possible competing work schedules after y_k.

Figure 2: Response-time analysis.

In the above expression, C_i accounts for the work required by J, overestimating by δ, while $U_i(y_k - t) + x_i U_i$ accounts for the work from all subsequent jobs (released after $t_0 - \delta - x_i + T_i - Y_i$), underestimating by $U_i\delta$ (so that our overestimation of J's work compensates). Observe that (7) is sufficient to upper-bound the remaining work even if J had its PP at or after t_0.

Case 3 can apply to at most $m-1$ tasks over $[t_0, t)$, because some CPU must be idle throughout $[t_0, t)$. Therefore, in light of (5), (6), and (7), we define

$$L(\vec{x}) = \sum_{m-1 \text{ largest}} x_i U_i + C_i - S_i \tag{8}$$

and

$$S(\tau) = \sum_{\tau_i \in \tau} S_i, \tag{9}$$

so that the remaining work at t is at most $L(\vec{x}) + S(\tau) + \sum_{\tau_i \in \tau} U_i(y_k - t)$. Then, because $m(y_k - t)$ units of work are completed in $[t, y_k)$ (see Fig. 2(a)), at most $L(\vec{x}) + S(\tau)$ units of work remain at y_k. In the worst case, J_k requires C_k units of execution, so the remaining jobs contribute at most $L(\vec{x}) + S(\tau) - C_k$. Observe that m processors are available to execute this competing work, all competing work is available after y_k, and J_k (or a predecessor) will execute after y_k whenever a processor is idle. Therefore, the situation is as depicted in Fig. 2(g), where the worst case is on the left. Thus, we can define

$$x_k = \frac{L(\vec{x}) + S(\tau) - C_k}{m} \tag{10}$$

so that (1) holds.

Observe that x_k is a component of $\vec{x}$ and therefore appears on both the right-hand and the left-hand sides of (10). As a result, we define a *minimum compliant vector* as an $\vec{x}$ that satisfies (10) for all k.

The minimum compliant vector can be computed in polynomial time, based on the observation that $L(\vec{x})$ and $S(\tau)$ are the same for all tasks, so we can denote

$$s = L(\vec{x}) + S(\tau) \tag{11}$$

and observe that, by (10) and (11), for all i,

$$x_i U_i + C_i - S_i = \frac{s - C_i}{m} \cdot U_i + C_i - S_i, \tag{12}$$

which is a linear equation with respect to s. Because $L(\vec{x})$ is defined as the $m-1$ largest such values and $S(\tau)$ is a constant, $L(\vec{x}) + S(\tau)$ is a piecewise linear function with respect to s. By finding its fixed point, we can then use (10) and (11) to compute each x_i value.

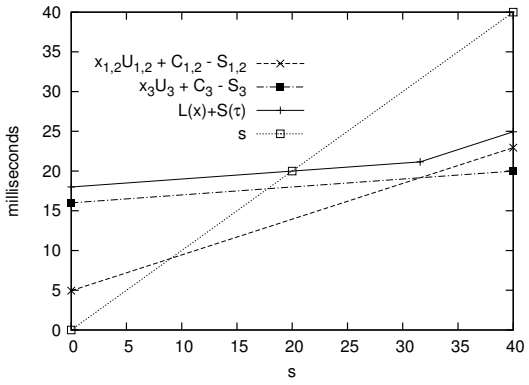


Figure 3: Compliant vector computation for θ.

Example. This situation is depicted for θ in Fig. 3, where we can see that $s = 20$ is the fixed point. Thus,

$$x_1 = \frac{20 - 9}{2} = 5.5,$$
$$x_2 = \frac{20 - 9}{2} = 5.5,$$
$$x_3 = \frac{20 - 20}{2} = 0.$$

Adding C_i for each task, we find that θ_1 and θ_2 have tardiness bounds of $5.5 + 9 = 14.5$, and θ_3 has a tardiness bound of $0 + 20 = 20$.

(1) provides a response-time bound for τ_i. Therefore, a tardiness bound for τ_i is given by

$$\max\{0, Y_i + x_i + C_i - D_i\}. \tag{13}$$

By (13), we can reduce the tardiness bound for task τ_i by decreasing Y_i. However, by doing so we increase the value of S_i by (4), and therefore the value of $S(\tau)$ by (9), affecting all tasks by (10) and (13). Thus, by decreasing Y_i we typically decrease the tardiness bound for τ_i, but typically increase the tardiness bounds for all other tasks in the system.

Example. Suppose we attempt to decrease the tardiness bound for θ_1 by using $Y_1 = 5$ (while still using $Y_2 = 10$ and $Y_3 = 90$). Then, by (4), $S_1 = 9(1 - \frac{5}{10}) = 4.5$. Therefore, $S(\tau) = 4.5 + 0 + 2 = 6.5$. Applying the algorithm of [5], we obtain $x_1 = 8$, $x_2 = 8$, and $x_3 = 2.5$, so by (13) we have tardiness bounds of 12, 17, and 22.5. Although we were successful in lowering the tardiness bound for θ_1, we decreased it by only 2.5 time units, and increased the tardiness bounds for both θ_2 and θ_3.

Having shown how to reinterpret prior analysis to apply to arbitrary GEL schedulers, we now show that

the *maximum* tardiness bound and observed tardiness can be decreased through a simple modification to G-EDF. Our approach to reducing tardiness is inspired by the EDZL scheduling algorithm. In EDZL, a task acquires the highest priority when it reaches a zero-laxity point. For our GEL algorithm, we attempt to achieve a similar benefit while statically allocating priorities to jobs. We assign PPs at the earliest time each job could achieve zero laxity, i.e., $Y_i = D_i - C_i$ (assuming that $D_i \geq C_i$). While we do not claim optimality for this choice of $\vec{Y}$, we provide experimental evidence demonstrating that it does typically improve maximum tardiness bounds and observed tardiness compared to G-EDF. These experiments provide strong evidence that G-EDF can be improved upon. In particular, the tardiness-bound experiments demonstrate that the work in Sec. 4 is also likely to improve upon G-EDF.[4]

To validate our improvements, we generated implicit-deadline task sets based on the experimental design from prior studies (e.g., [13]). We generated task utilizations using either a uniform or a bimodal distribution. For task sets with uniformly distributed utilizations, we used either a *light* distribution with values randomly chosen from $[0.001, 0.1]$, a *medium* distribution with values randomly chosen from $[0.1, 0.4]$, or a *heavy* distribution with values randomly chosen from $[0.5, 0.9]$. For tasks sets with bimodally distributed utilizations, values were chosen uniformly from either $[0.001, 0.5]$ or $[0.5, 0.9]$, with respective probabilities of $8/9$ and $1/9$ for *light* distributions, $6/9$ and $3/9$ for *medium* distributions, and $4/9$ and $5/9$ for *heavy* distributions. We generated integral task periods using a uniform distribution from $[3ms, 33ms]$ for *short* periods, $[10ms, 100ms]$ for *moderate* periods, or $[50ms, 250ms]$ for *long* periods. Each experiment was run with either $m = 2$, $m = 4$, or $m = 6$ CPUs. By [7] we do not need to consider significantly larger numbers of CPUs, as clustered scheduling will outperform global scheduling for such systems in practice.

We ran two experiments for each possible combination of utilization distribution, period distribution, and processor count: one to determine the effect of our EDZL-inspired modification on tardiness bounds, and one to determine its effect on observed tardiness. For each experiment, we generated 1000 task sets. For each task set τ, we generated tasks until generating one that would cause $\sum_{\tau_i \in \tau} U_i$ to exceed m. For observed-tardiness experiments, we restricted task worst-case execution times to be integers.

For each task set, we computed "maximum tardiness" metrics g for G-EDF and h for the GEL scheduler with $Y_i = D_i - C_i$. In the tardiness-bound experiments, g and h were defined as the maximum tardiness bounds computed using the methods described here.[5] In the observed-tardiness experiments, we defined g and h as the maximum tardiness experienced by any job during the first 100 seconds of execution, when all job releases are periodic and all jobs run for their full worst-case execution times. For each experiment, we computed $\bar{g}$, the mean g value over all task sets, and $\bar{h}$, the mean h value over all task sets. The *relative improvement* for an experiment is defined as $(\bar{g} - \bar{h})/\bar{g}$.

Results of our experiments are shown in Fig. 4. Tardiness-bound experiments often showed an improvement of about 30%, and observed-tardiness experiments sometimes showed an improvement exceeding 99%. These results demonstrate that G-EDF can be significantly improved upon within the class of GEL schedulers, both in terms of bounds achievable by the designer and actual observed tardiness.

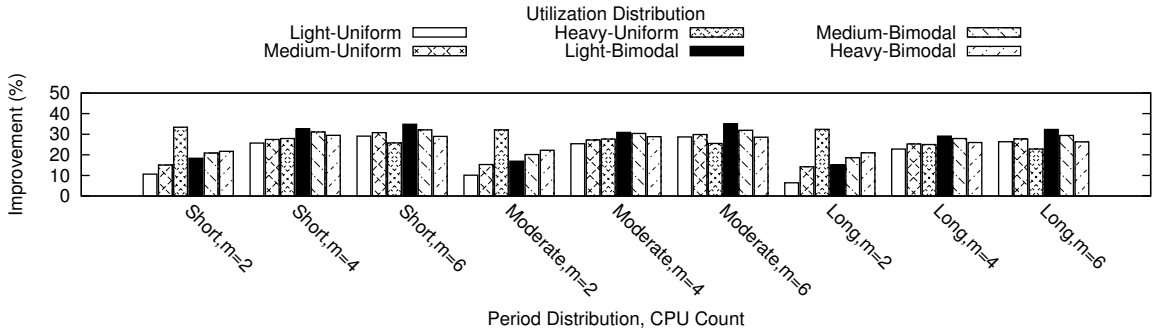
4 Algorithm for Meeting Desired Response-Time Bounds

In Sec. 3.2 above, we described a method to produce either response-time bounds (1) or tardiness bounds (13) for GEL schedulers given task system parameters and a $\vec{Y}$. However, in practice, a system designer is likely to have a desired set of response-time bounds, and requires a $\vec{Y}$ to achieve those bounds. For HRT systems, these response-time bounds are simply relative deadlines. For SRT systems, these response-time bounds indicate relative deadlines plus an additional term for acceptable tardiness. In either case, we will denote the response-time bound for task τ_i as R_i. We define $\vec{R} = \langle R_1, R_2, \ldots, R_n \rangle$. We provide a polynomial-time algorithm that determines whether the bounds specified by $\vec{R}$ can be achieved for task system τ (using compliant-vector analysis), and if so, returns a $\vec{Y}$ sufficient for achieving those bounds. Because individual R_i values can indicate either HRT or SRT requirements, both requirements can be mixed in the same system.

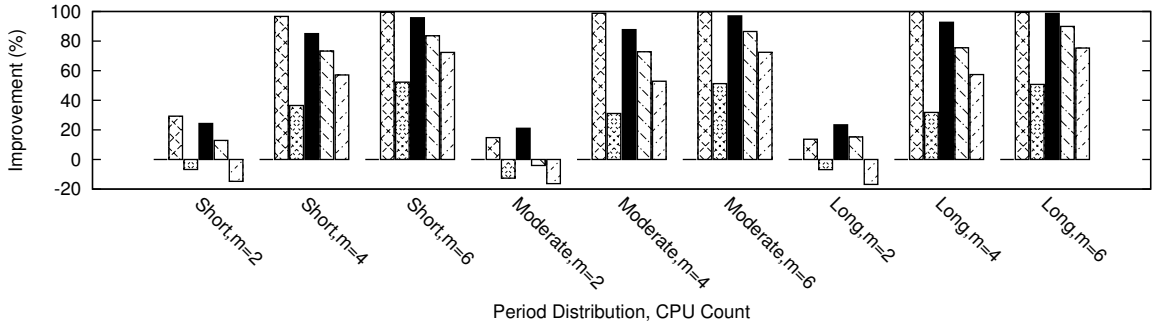
In Sec. 4.1, we provide theoretical foundations necessary to understand our algorithm, and in Sec. 4.2, we describe the algorithm itself.

[4] This is true because the experiments demonstrate that, for each task set, a $\vec{Y}$ exists providing a lower maximum tardiness bound than G-EDF. The algorithm in Sec. 4 is guaranteed to find such a $\vec{Y}$ if the designer specifies compatible response-time bounds.

[5] Because our task systems were implicit-deadline, an improvement mentioned in [4] was possible in the G-EDF case. However, we wanted our results to represent arbitrary-deadline systems, so we used the best method known for arbitrary-deadline systems.



(a) Relative Improvement in Tardiness Bounds.



(b) Relative Improvement for Observed Tardiness.

Figure 4: Improvement by using $Y_i = D_i - C_i$ instead of $Y_i = D_i$ for representative implicit-deadline task systems.

4.1 Theoretical Foundations

Observe that for each τ_i, by (1),

$$R_i = Y_i + x_i + C_i. \tag{14}$$

Prior analysis provides insight into the computation of x_i. In the present analysis, R_i and C_i are constants, and our goal is to calculate Y_i. Therefore, we rearrange (14) as follows:

$$Y_i = R_i - x_i - C_i. \tag{15}$$

Because by (4) S_i depends on Y_i, S_i now depends on x_i. Therefore, we rename it as $S_i(x_i)$ to reflect this dependency. We now have:

$$\begin{aligned} S_i(x_i) &= \{\text{By (4) and (15)}\} \\ &\quad \max\left\{0, C_i\left(1 - \frac{R_i - x_i - C_i}{T_i}\right)\right\} \qquad (16) \\ &= \{\text{Rearranging}\} \\ &\quad \max\{0, C_i - (R_i - C_i)U_i + x_i U_i\}. \qquad (17) \end{aligned}$$

Because $S_i(x_i)$ now depends on x_i, by (9), $S(\tau)$ now depends on $\vec{x}$, whereas in Sec. 3.2, $S(\tau)$ was based only on task system parameters and $\vec{Y}$. We therefore rename $S(\tau)$ to $S(\vec{x})$ by defining

$$S(\vec{x}) = \sum_{\tau_i \in \tau} S_i(x_i). \tag{18}$$

We also update the definition of $L(\vec{x})$ from (8) to reflect the same change

$$L(\vec{x}) = \sum_{m-1 \text{ largest}} x_i U_i + C_i - S_i(x_i) \tag{19}$$

and modify (10) to use for each τ_i the constraint

$$x_i = \frac{L(\vec{x}) + S(\vec{x}) - C_i}{m}. \tag{20}$$

We say that a vector conforming to (17)–(20) is a *minimal compliant vector* with respect to τ and $\vec{R}$.

Property 1. *$\vec{x}$ is a minimal compliant vector with respect to τ and $\vec{R}$ iff the corresponding $\vec{Y}$ (defined by (15)), τ, and $\vec{x}$ satisfy (4), (8), (9), and (10).*

Prop. 1 holds by our construction of (17)–(20). However, despite Prop. 1, it is possible that the $\vec{Y}$ defined by (15) has some $Y_i < 0$, and therefore violates the conditions of the proof sketched in Sec. 3.2. Therefore, we define a $\vec{Y}$ as *valid* iff for all i, $Y_i \geq 0$.

Property 2. *$\vec{x}$ is a minimal compliant vector with respect to τ and $\vec{R}$, and the corresponding $\vec{Y}$ defined by (15) is valid, iff $\vec{x}$ is a* minimum *compliant vector with respect to τ and $\vec{Y}$.*

Prop. 2 holds by Prop. 1, because the validity of $\vec{Y}$ is necessary and sufficient to ensure that the analysis in Sec. 3.2 applies. By Prop. 2 and (14), we have Prop. 3 and Prop. 4.

Property 3. *If $\vec{x}$ is a minimal compliant vector with respect to τ and $\vec{R}$, and the corresponding $\vec{Y}$ defined by (15) is valid, then $\vec{Y}$ satisfies response-time bounds $\vec{R}$.*

Property 4. *If there is a $\vec{Y}$ satisfying response-time bounds $\vec{R}$ using compliant-vector analysis, then the corresponding $\vec{x}$ implied by (15) is a minimal compliant vector with respect to τ and $\vec{R}$.*

We now work our way to Thm. 1, which provides the mathematical foundation for our algorithm to determine whether a valid $\vec{Y}$ satisfying response-time bounds $\vec{R}$ can be computed and, if so, to compute it.

Lemma 1. *There is a real number s such that*

$$s = L(\vec{x}) + S(\vec{x}), \tag{21}$$

and for each τ_i

$$x_i = \frac{s - C_i}{m}, \tag{22}$$

iff $\vec{x}$ is a minimal compliant vector with respect to τ and $\vec{R}$.

Proof. Suppose $\vec{x}$ is a minimal compliant vector with respect to τ and $\vec{R}$. Define s such that (21) holds. Then, by (20), (22) holds.

Suppose there is some s such that (21) and (22) hold. Then, by (21), (20) holds. (17)–(19) are true by definition, so $\vec{x}$ is a minimal compliant vector with respect to τ and $\vec{R}$. □

Our algorithm will attempt to determine such a value of s. To do so, we define a few functions.

$$\vec{v}(s) = \vec{x} \text{ such that (22) holds} \tag{23}$$
$$L(s) = L(\vec{v}(s)) \tag{24}$$
$$S(s) = S(\vec{v}(s)) \tag{25}$$
$$M(s) = L(s) + S(s) - s \tag{26}$$

Most of our analysis will apply to $\vec{v}(s)$ in place of $\vec{x}$. For convenience, we rewrite (15)–(20) with this substitution.

$$Y_i = R_i - v_i(s) - C_i \tag{27}$$

$$S_i(v_i(s)) = \max\left\{0, C_i\left(1 - \frac{R_i - v_i(s) - C_i}{T_i}\right)\right\} \tag{28}$$
$$= \max\{0, C_i - (R_i - C_i)U_i + v_i(s)U_i\} \tag{29}$$
$$S(\vec{v}(s)) = \sum_{\tau_i \in \tau} S_i(v_i(s)) \tag{30}$$
$$L(\vec{v}(s)) = \sum_{m-1 \text{ largest}} \vec{v}(s)U_i + C_i - S_i(v_i) \tag{31}$$
$$v_i(s) = \frac{L(\vec{v}(s)) + S(\vec{v}(s)) - C_i}{m}. \tag{32}$$

Furthermore, we rewrite Lem. 1 with the same change in notation, as Cor. 1.

Corollary 1. *There is a real number s such that*

$$s = L(\vec{v}(s)) + S(\vec{v}(s)) \tag{33}$$

iff $\vec{v}(s)$ is a minimal compliant vector with respect to τ and R.

The following lemma utilizes these notations to characterize the desired minimal compliant vector.

Lemma 2. *$M(s) = 0$ iff $\vec{v}(s)$ is a minimal compliant vector with respect to τ and $\vec{R}$.*

Proof. Suppose $M(s) = 0$. Then by (24)–(26), $s = L(\vec{v}(s)) + S(\vec{v}(s))$, so (33) holds. Therefore, by Cor. 1, $\vec{v}(s)$ is a minimal compliant vector with respect to τ and $\vec{R}$.

Suppose $\vec{v}(s)$ is a minimal compliant vector with respect to τ and $\vec{R}$. Then by Cor. 1, (33) holds. Therefore, by (24)–(26), $M(s) = 0$. □

In the following, we concern ourselves with an interval $[s_{\min}, s_{\max}]$, where

$$s_{\min} = \max_{\tau_i \in \tau}(C_i), \tag{34}$$
$$s_{\max} = \min_{\tau_i \in \tau}(C_i + m(R_i - C_i)). \tag{35}$$

These definitions will allow us to prove Thm. 1, which demonstrates that s must fall in $[s_{\min}, s_{\max}]$ in order to meet the response-time bounds specified by $\vec{R}$.

The next two lemmas are proved in an appendix.

Lemma 3. *$s < s_{\min}$ implies $M(s) > 0$.*

Lemma 4. *There exists an i such that $R_i - v_i(s) - C_i < 0$ iff $s > s_{\max}$.*

Theorem 1. *There exists a valid $\vec{Y}$ satisfying response-time bounds $\vec{R}$ for task system τ (using compliant-vector analysis) iff there is an s in $[s_{\min}, s_{\max}]$ such that $M(s) = 0$.*

Proof. Suppose there is an s in $[s_{\min}, s_{\max}]$ such that $M(s) = 0$. Then by Lem. 2, $\vec{v}(s)$ is a minimal compliant vector with respect to τ and $\vec{R}$. Let $\vec{Y}$ be as in (27). Because $s \leq s_{\max}$, by (27) and Lem. 4, for all i, $Y_i \geq 0$. Therefore, Prop. 3 ensures that $\vec{Y}$ will satisfy response-time bounds $\vec{R}$.

Suppose there exists a valid $\vec{Y}$ satisfying response-time bounds $\vec{R}$ (using compliant-vector analysis). Then, by Prop. 4, there is a minimal compliant vector $\vec{x}$ with respect to τ and $\vec{R}$ that satisfies (15). By Lem. 1, (22), and (23), there exists an s such that $\vec{x} = \vec{v}(s)$, and by Lem. 2, $M(s) = 0$. Therefore, by Lem. 3, $s \geq s_{\min}$. By the validity of $\vec{Y}$, (27), and Lem. 4, $s \leq s_{\max}$. □

We prove two theorems in the appendix that demonstrate the flexibility of our algorithm. Here we list and briefly describe them.

Applied inductively, the first result demonstrates that a system designer can safely use conservatively large R_i values, and this will not compromise the ability to obtain a valid $\vec{Y}$.

Theorem 2. *Suppose there exists a valid $\vec{Y}$ satisfying response-time bounds $\vec{R}$ using compliant-vector analysis. Consider $\vec{R}'$ such that there is an i such that $R'_i > R_i$, but for all $j \neq i$, $R'_j = R_j$. Then, there exists a valid $\vec{Y}'$ satisfying response-time bounds $\vec{R}$.*

The second result demonstrates that using this approach, the maximum useful choice of Y_i is T_i, and using larger Y_i values will unnecessarily increase R_i bounds. Therefore, when the algorithm returns some $Y_i > T_i$, the designer can decrease R_i and continue to meet all other bounds.

Theorem 3. *If valid $\vec{Y}$ satisfies response-time bounds $\vec{R}$, and for some i, $Y_i > T_i$, then valid $\vec{Y}'$ such that $Y'_i = T_i$ and, for $j \neq i$, $Y'_j = Y_j$, satisfies response-time bounds $\vec{R}'$ such that $R'_i = R_i - (Y_i - T_i)$ and, for $j \neq i$, $R'_j = R_j$.*

4.2 Algorithm

We now present an algorithm that will determine whether it is possible to meet response-time bounds $\vec{R}$, and if so, to compute the $\vec{Y}$ necessary to do so. This algorithm works by tracking the value of $M(s)$ to determine the smallest s in $[s_{\min}, s_{\max}]$ such that $M(s) = 0$. It is similar to an algorithm provided in [5] in that both algorithms track the value of a piecewise linear function to find the desired value. First, denote (for each i)

$$h_i = m(R_i - C_i - T_i) + C_i. \tag{36}$$

Then, by (22), (23), (29) and (36),

$$S_i(v_i(s)) = \begin{cases} 0 & \text{if } s < h_i \\ C_i - (R_i - C_i)U_i + \frac{s - C_i}{m} \cdot U_i & \text{if } s \geq h_i. \end{cases} \tag{37}$$

For convenience, we denote

$$l_i(s) = v_i(s)U_i + C_i - S_i(v_i(s)). \tag{38}$$

Therefore, by (22), (23), (37), and (38),

$$l_i(s) = \begin{cases} \frac{s - C_i}{m} \cdot U_i + C_i & \text{if } s < h_i \\ (R_i - C_i)U_i & \text{if } s \geq h_i. \end{cases} \tag{39}$$

By (25), (30), and (37), $S(s)$ changes slope at each h_i, or $O(n)$ times in total. By (24), (31), (38), and (39), $L(s)$ may change slope in either of two conditions.

- $L(s)$ may change slope at an h_i value. This may occur once per task, or $O(n)$ times in total.
- $L(s)$ may change slope when $l_i(s) = l_j(s)$ for some $i \neq j$ (due to the "$m-1$ largest" condition in (31)). This may occur at most $O(n^2)$ times in total, and each i, j pair can be checked in constant time.

Like the algorithm in [5], our algorithm works by sorting the list of points where the slope may change, and examining the linear component between two consectuive points. Therefore, it still has complexity $O(n^2 \log n + mn^2)$, because we add only $O(n)$ additional points where the slope can change.

Example. To demonstrate our algorithm, we will consider an attempt to schedule θ with a mix of requirements. θ_1 will be scheduled as an SRT task where a maximum tardiness of 19 is acceptable, θ_2 will be scheduled as an SRT task where arbitrary tardiness is acceptable, and θ_3 will be scheduled as an HRT task. By Thm. 2, the designer can select a conservatively large value to meet the requirement for θ_2. Therefore, we will use $R_1 = 29$, $R_2 = 99$, and $R_3 = 90$. By (34), $s_{\min} = 20$, and by (35), $s_{\max} = 49$.

We graph each $S_i(v_i(s))$ and $l_i(s)$ from $s = 0$ to $s = 50$, along with the resulting $L(s)+S(s)$ and s, in Fig. 5. (Observe that by (26), $M(s) = 0$ iff $L(s) + S(s) = s$.) The fixed point of $L(s) + S(s)$ occurs at $s = 20$, which is within the range $[s_{\min}, s_{\max}]$, so a valid solution does exist. Specifically, by (15), (22), and (23),

$$Y_1 = 29 - \frac{20 - 9}{2} - 9 \quad = 14.5$$

$$Y_2 = 99 - \frac{20 - 9}{2} - 9 \quad = 84.5$$

$$Y_3 = 90 - \frac{20 - 20}{2} - 20 = 70$$

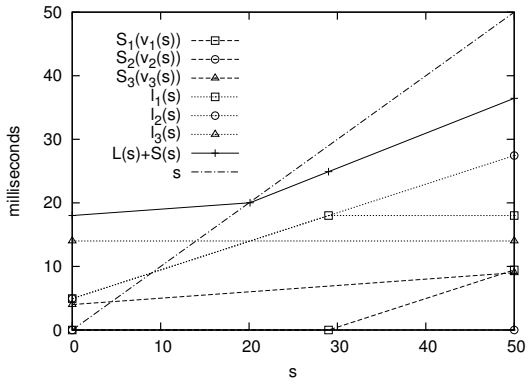


Figure 5: Functions used by our algorithm on θ.

Because $Y_1 > T_1$ and $Y_2 > T_2$, by Thm. 3 (applied twice), (22), and (23), we can set $Y_1' = 10$ and $Y_2' = 10$ to obtain

$$\begin{aligned} R_1' &= 29 - (14.5 - 10) = 24.5 \\ R_2' &= 99 - (84.5 - 10) = 24.5 \\ R_3' &= 90 \qquad\qquad\quad = 90, \end{aligned}$$

which are better than specified.

5 Conclusion

We have demonstrated that G-EDF can be improved upon within the class of GEL schedulers, whether supporting HRT, SRT, or mixed workloads. In the case of HRT scheduling, we have provided proof that there are task systems schedulable with GEL schedulers, but not with G-EDF. In the case of SRT scheduling, we have provided experimental evidence that maximum tardiness can be reduced, both in terms of tardiness bounds and in terms of observed tardiness. Notably, we achieved these improvements without sacrificing the JLSP property of G-EDF, and thus existing work on real-time synchronization can be used with our proposals. Future work could explore whether other practical JLSP schedulers are possible that improve upon GEL schedulers.

We have also provided an algorithm to determine PPs for GEL schedulers to ensure meeting specific response-time bounds. Future work could explore the possibility of gaining benefit from allowing $Y_i < 0$ or taking advantage of $Y_i > T_i$. Furthermore, specific proposals, such as the $Y_i = D_i - C_i$ algorithm explored in Sec. 3.2, could be explored in an HRT context, and schedulabilty tests beyond the response-time bounds provided here could be explored.

References

[1] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, January 1973.

[2] M. Bertogna and S. Baruah, "Tests for global EDF schedulability analysis," *Journal of Systems Architecture*, to appear.

[3] U. C. Devi and J. H. Anderson, "Tardiness bounds under global EDF scheduling on a multiprocessor," *The Journal of Real-Time Systems*, vol. 38, no. 2, pp. 133–189, 2008.

[4] J. P. Erickson, U. Devi, and S. K. Baruah, "Improved tardiness bounds for global EDF," in *ECRTS*, 2010, pp. 14–23.

[5] J. P. Erickson, N. Guan, and S. K. Baruah, "Tardiness bounds for global EDF with deadlines different from periods," in *OPODIS*, 2010, pp. 286–301.

[6] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, pp. 600–625, 1996.

[7] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers," in *RTSS*, 2010, pp. 14–24.

[8] H. Cho, B. Ravindran, and E. D. Jensen, "Efficient real-time scheduling algorithms for multiprocessor systems," *IEICE Trans. Communications*, vol. 85, pp. 807–813, 2002.

[9] T. Baker, M. Cirinei, and M. Bertogna, "EDZL scheduling analysis," *Real-Time Systems*, vol. 40, pp. 264–289, 2008.

[10] H. Leontyev and J. H. Anderson, "Generalized tardiness bounds for global multiprocessor scheduling," *The Journal of Real-Time Systems*, vol. 44, no. 1, pp. 26–71, February 2010.

[11] H. Leontyev, S. Chakraborty, and J. H. Anderson, "Multiprocessor extensions to real-time calculus," in *RTSS*, 2009, pp. 410–421.

[12] J. Lee, A. Easwaran, and I. Shin, "Maximizing contention-free executions in multiprocessor scheduling," in *RTAS*, 2011, pp. 235–244.

[13] A. Bastoni, B. B. Brandenburg, and J. H. Anderson, "Is semi-partitioned scheduling practical?" in *ECRTS*, 2011, to appear.

Appendix

Here we prove several lemmas and theorems stated in Sec. 4.1.

Lemma 3. *$s < s_{\min}$ implies $M(s) > 0$.*

Proof. Let j be such that $C_j = s_{\min}$, which must be possible by (34). Let τ' be the set of the $m-2$ tasks τ_i ($i \neq j$) with the largest values of $v_i(s)U_i + C_i$. Suppose $s < s_{\min}$. Then,

$$s < C_j. \tag{40}$$

Therefore,

$$\begin{aligned}
M(s) &= \{\text{By (26)}\} \\
&\quad L(s) + S(s) - s \\
&= \{\text{By (22)–(25), (29), and (31)}\} \\
&\quad \sum_{m-1 \text{ largest}} \left(\frac{s - C_i}{m} U_i + C_i - S_i\left(\frac{s - C_i}{m}\right)\right) \\
&\quad + \sum_{\tau_i \in \tau} S_i\left(\frac{s - C_i}{m}\right) - s \\
&\geq \{\text{Because each } S_i(\tfrac{s-C_i}{m}) \geq 0, \text{ by (29); note} \\
&\quad \text{that each } S_i(s - C_i/m) \text{ appearing in the} \\
&\quad L(s) \text{ summation also appears in the } S(s) \\
&\quad \text{summation}\} \\
&\quad \sum_{m-1 \text{ largest}} \left(\frac{s - C_i}{m} U_i + C_i \right) - s \\
&\geq \{\text{By definition of ``largest''; recall that } \tau' \\
&\quad \text{includes } m-2 \text{ tasks}\} \\
&\quad \frac{s - C_j}{m} U_j + C_j \\
&\quad + \sum_{\tau_i \in \tau'} \left(\frac{s - C_i}{m} U_i + C_i \right) - s \\
&= \{\text{Rearranging}\} \\
&\quad \frac{m - U_j}{m} C_j + \sum_{\tau_i \in \tau'} \left(\frac{m - U_i}{m} C_i \right) \\
&\quad - \frac{m - \left(U_j + \sum_{\tau_i \in \tau'} U_i\right)}{m} s \\
&> \{\text{Because for all } i,\ \tfrac{m-U_i}{m} C_i > 0\} \\
&\quad \frac{m - U_j}{m} C_j - \frac{m - \left(U_j + \sum_{\tau_i \in \tau'} U_i\right)}{m} s \\
&> \{\text{By (40)}\} \\
&\quad 0.
\end{aligned}$$

□

Lemma 4. *There exists an i such that $R_i - v_i(s) - C_i < 0$ iff $s > s_{\max}$.*

Proof. Suppose $s > s_{\max}$. Then by (35) there is an i such that

$$s > C_i + m(R_i - C_i). \tag{41}$$

Therefore,

$$\begin{aligned}
R_i - v_i(s) - C_i &= \{\text{By (22) and (23)}\} \\
&\quad R_i - \frac{s - C_i}{m} - C_i \\
&< \{\text{By (41)}\} \\
&\quad R_i - \frac{C_i + m(R_i - C_i) - C_i}{m} - C_i \\
&= \{\text{Simplifying}\} \\
&\quad 0.
\end{aligned}$$

Suppose there is an i such that $R_i - v_i(s) - C_i < 0$. Then, by (22) and (23),

$$\begin{aligned}
& R_i - \frac{s - C_i}{m} - C_i < 0 \\
\Rightarrow & \frac{s}{m} > R_i + \frac{C_i}{m} - C_i \\
\Rightarrow & s \ > C_i + m(R_i - C_i).
\end{aligned}$$

Therefore, by (35), $s > s_{\max}$. □

In both of the following theorems, we define an $\vec{R}'$, and denote all operations and variables with respect to τ and $\vec{R}'$ with a prime: $M'(s)$, $S'(v'_i(s))$, $s'_{\min}$, etc. Observe from (22) and (23) that

$$v'_i(s) = v_i(s) = \frac{s - C_i}{m}. \tag{42}$$

Theorem 2. *Suppose there exists a valid $\vec{Y}$ satisfying response-time bounds $\vec{R}$ using compliant-vector analysis. Consider $\vec{R}'$ such that there is an i such that $R'_i > R_i$, but for all $j \neq i$, $R'_j = R_j$. Then, there exists a valid $\vec{Y}'$ satisfying response-time bounds $\vec{R}$.*

Proof. By (27) and (29)–(32), any increase to some R_i can potentially decrease Y_i or some Y_j only by increasing $L(\vec{v}(s))$, which can occur if $S_i(v_i(s))$ decreases. For $j \neq i$, by (29) and (42), $S'_j(v'_j(s)) = S_j(v_j(s))$. For i,

$$\begin{aligned}
S'_i(v'_i(s)) &= \{\text{By (29)}\} \\
&\quad \max\{0, C_i - (R'_i - C_i)U_i + v'_i(s)U_i\} \\
&\leq \{\text{Because } R'_i > R_i \text{ and by (42)}\} \\
&\quad \max\{0, C_i - (R_i - C_i)U_i + v_i(s)U_i\} \\
&= (\text{By (29)}\} \\
&\quad S_i(v_i(s))
\end{aligned}$$

. We will use the above results to show that $M'(s) \leq M(s)$.

$M'(s) = \{\text{By (24)–(26), (30), and (31)}\}$

$$\sum_{m-1 \text{ largest}} (v'_k(s)U_k + C_k - S'_k(v'_k(s))) + \sum_{\tau_k \in \tau} S'_k(v'_k(s)) - s$$

$\leq$ {Because $S'_k(v'_k(s)) \leq S_k(v_k(s))$ and by (42); note that each $S'_k(v'_k(s))$ appearing in the $L(s)$ summation also appears in the $S(s)$ summation}

$$\sum_{m-1 \text{ largest}} (v_i(s)U_k + C_k - S_k(v_k(s))) + \sum_{\tau_k \in \tau} S_k(v_k(s)) - s$$

$=$ {By (24)–(26), (30), and (31)}

$M(s)$

By Thm. 1, $M(s) = 0$, so $M'(s) \leq 0$. Observe that $L'(s)$ and $S'(s)$ are piecewise linear with respect to s by (22)–(25) and (29)–(31). Therefore, by (26), $M'(s)$ is piecewise linear and therefore continuous. Let $\epsilon > 0$. Then, by Lem. 3, $M'(s'_{\min} - \epsilon) > 0$. Therefore, because ϵ is arbitrary, by the Intermediate Value Theorem there must exist $s' \in [s'_{\min}, s]$ such that $M'(s') = 0$.

By (35), $s_{\max} \leq s'_{\max}$. Therefore, $s \in [s'_{\min}, s'_{\max}]$. By Thm. 1, there exists a valid $\vec{Y}'$ satisfying response-time bounds $\vec{R}'$. □

Theorem 3. *If valid $\vec{Y}$ satisfies response-time bounds $\vec{R}$, and for some i, $Y_i > T_i$, then valid $\vec{Y}'$ such that $Y'_i = T_i$ and, for $j \neq i$, $Y'_j = Y_j$, satisfies response-time bounds $\vec{R}'$ such that $R'_i = R_i - (Y_i - T_i)$ and, for $j \neq i$, $R'_j = R_j$.*

Proof. By Thm 1, there exists an s in $[s_{\min}, s_{\max}]$ such that $M(s) = 0$; thus, by Lem. 2, $\vec{v}(s)$ is a minimal compliant vector with respect to τ and $\vec{R}$. Let $\vec{Y}$ be as in (27).

$$\begin{aligned} S_i(v_i(s)) &= \{\text{By (27) and (28)}\} \\ &\quad \max\left\{0, C_i\left(1 - \frac{Y_i}{T_i}\right)\right\} \\ &= \{\text{Because } Y_i > T_i\} \\ &\quad 0 \end{aligned}$$

Let $R'_i = R_i - (Y_i - T_i)$, so that by (27),

$$R'_i - v_i(s) - C_i = T_i. \tag{43}$$

We now have

$$\begin{aligned} S'_i(v'_i(s)) &= \{\text{By (42)}\} \\ &\quad S'_i(v_i(s)) \\ &= \{\text{By (28) and (43)}\} \\ &\quad 0, \end{aligned}$$

and therefore $S_i(v_i(s)) = S'_i(v'_i(s))$. By (27), (29) and (30)–(32), changing R'_i therefore has no effect on $S(\vec{v}(s))$ or $L(\vec{v}(s)$. Therefore, it also has no effect on R_j or Y_j for $j \neq i$. Furthermore, we defined $Y'_i = T_i > 0$, so the resulting $\vec{Y}'$ must be valid. □