CHAPTER 4

Removing Intra-Task Precedence Constraints

In this chapter, we provide response-time bounds for a class of workloads in which jobs of the same task are independent of each other and can be executed in parallel, such as servers handling independent requests. In addition to providing analysis that applies to arbitrary GEL schedulers, we show that lateness 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 lateness bounds can be deduced. After deriving such bounds, we compare them experimentally to prior bounds, which were derived assuming no intra-task parallelism.

The remainder of this chapter is organized as follows. In Section 4.1, we formally present the task model assumed in this chapter. Then, in Section 4.2, we describe compliant vectors for this task model, like those described in Chapter 3. Afterward, in Section 4.3, we prove that each task system has a unique minimum compliant vector, as was the case in Chapter 3. We then provide in Section 4.4 an algorithm to compute the minimum compliant vector, and in Section 4.5, we provide experimental evidence that removing precedence constraints can significantly reduce lateness bounds. We conclude in Section 4.6.

4.1 System Model

In the task model discussed in Section 1.1.1, successive jobs of each task were required to execute in sequence. This constraint arises naturally when jobs correspond to separate invocations of the same code segment, as discussed in Section 1.1.1. However, in some settings, jobs are released as separate threads in response to interrupts, in which case, successive jobs of the same task may

---

1This work appeared in preliminary form in the following paper:
execute concurrently. In prior hard real-time analysis of G-EDF by Baker and Baruah (2009), the impact of such concurrently executing jobs has been considered, but to our knowledge, no such analysis exists for SRT systems for which bounded lateness is acceptable.

Compared to the task model described in Section 1.1.1, we make two modifications. First, as implied by the discussion above, successive jobs of the same task are allowed to execute in parallel. Second, *early release* behavior (Anderson and Srinivasan, 2000) 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 and PP are defined based on its s-release time, and consecutive s-releases of each task \( \tau_i \) are still constrained to be no closer than \( T_i \) time units apart. 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 (Durbin, 2010). 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, even on a multiprocessor. However, it can be supported using the methodology provided in this chapter. 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 under G-EDF \((Y_i = D_i)\) on two processors is depicted in Figure 4.1. 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.

![Figure 4.1: Example high-frequency trading system scheduled with G-EDF.](image)
In the task model considered here, no job may run concurrently with itself, but distinct jobs within the same task may run concurrently. For convenience, we define the task system utilization \[ U(\tau) \triangleq \sum_{\tau_i \in \tau} U_i, \] and \[ U^+ \triangleq \lceil U(\tau) \rceil. \]

Under the task model described in Section 1.1.1 with implicit precedence constraints, providing bounded response time required that no \( \tau_i \) had \( U_i > 1 \) and that \( U(\tau) \leq m \), as in Chapter 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 \( \tau_1 \) in Figure 4.1. \( U(\tau) \leq m \) 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 relative to the s-release time of each job.

### 4.2 Response Time Characterization

Over an interval of any given length \( \Delta \), the total amount of work from jobs of \( \tau_i \) (with both s-release times and PPs inside the interval) is bounded. The same demand-bound function \( DBF(\tau_i, Y_i, \Delta) \) used in Chapter 3 continues to apply to our broader task model, as long as we consider the release time of each job to be its s-release. If a job actually has an a-release prior to the beginning of an interval but an s-release within an interval, that can only reduce the demand within the interval compared to the situation in which the job’s a-release time was equal to its s-release time.

We will use Lemma 3.2 to provide an upper bound on \( DBF(\tau_i, Y_i, \Delta) \). We state that lemma again here for convenience. (Recall that, by (3.8), \( S_i(Y_i) \triangleq C_i \cdot \max\{0, 1 - Y_i/T_i\} \).)

**Lemma 3.2.** \( \forall \Delta \geq 0, DBF(\tau_i, Y_i, \Delta) \leq U_i \Delta + S_i(Y_i) \).

For an \( n \)-task system \( \tau \), 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 \( \tau_i \), \( 1 \leq i \leq n \), is at most \( x_i + C_i \) when \( \tau \) is scheduled using a GEL scheduler 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 Chapter 3 we define the notion of a “compliant vector” as one that meets this criterion.
Definition 4.1. For each task \( \tau_i \), non-negative integer \( p < U^+ - 1 \), and non-negative real number \( x_i \), let

\[
g(\tau_i, x_i, p) \triangleq \min \{ C_i, \max \{ 0, x_i + C_i - pT_i \} \}.
\]

(4.1)

For any \( \vec{x} \triangleq (x_1, x_2, \ldots, x_n) \), an ordered list of \( n \) non-negative real numbers, let

\[
G(\vec{x}) \triangleq \sum_{U^+ - 1 \text{ largest}} g(\tau_i, x_i, p),
\]

(4.2)

\[
S(\vec{Y}) \triangleq \sum_{\tau_i \in \tau} S_i(Y_i).
\]

(4.3)

We define \( \vec{x} \) as a compliant vector if and only if

\[
\frac{G(\vec{x}) + S(\vec{Y}) + U(\tau)Y_i - C_i}{m} \leq x_i
\]

(4.4)

is satisfied for all \( i, 1 \leq i \leq n \).

Observe that, unlike \( G(\vec{x}, \vec{Y}) \) as defined in (3.12), \( G(\vec{x}) \) as defined in (4.2) does not depend on \( \vec{Y} \). However, the definition of \( S(\vec{Y}) \) in (4.3) is identical to that in (3.9). As compared to (3.13), in (4.4) we also add the term \( U(\tau)Y_i \) in the numerator. This term is necessary because we are considering response-time bounds instead of lateness bounds.

We now derive a response-time bound by considering a compliant vector \( \vec{x} = (x_1, x_2, \ldots, x_n) \) and an arbitrary collection \( H' \) of jobs generated by \( \tau \). We order jobs by PP with ties broken arbitrarily (as per standard GEL scheduling). We analyze the response time of an arbitrary job \( \tau_{i,k} \) with s-release time \( r_{i,k} \) and PP \( y_{i,k} \), assuming that each \( \tau_{j,\ell} \) ordered prior to \( \tau_{i,k} \) completes within \( (C_j + x_j) \) units of its s-release time. We denote as \( H \) the set of all jobs ordered at or before \( \tau_{i,k} \), which (by the definition of GEL scheduling) contains all jobs that affect the scheduling of \( \tau_{i,k} \). We also denote \( H_c \triangleq H \setminus \{ \tau_{i,k} \} \) (i.e., the work competing with \( \tau_{i,k} \)).

We denote as \( W_j(t) \) the remaining execution for jobs in \( H \) of task \( \tau_j \) at time \( t \), and let \( W(t) \triangleq \sum_{\tau_j \in \tau} W_j(t) \). Furthermore, we define an interval as busy if at least \( U^+ \) processors are executing work throughout the interval, and nonbusy otherwise. We define \( t_b \) (busy) as the earliest time such that \( [t_b, r_{i,k}) \) is continuously busy. At the latest, \( t_b = r_{i,k} \), in which case the interval is empty. Observe that, unlike in Chapter 3, the busy interval ends at \( r_{i,k} \) rather than \( y_{i,k} \).
In Lemma 4.1 below, we will bound $W(t_b)$. Then, in Lemma 4.2 below, we will use that result to bound $W(r_{i,k})$. Then, in Lemma 4.3 we will provide a response-time bound for $\tau_{i,k}$, and in Theorem 4.1, we will prove by induction that this bound is correct for all $\tau_{i,k}$.

**Lemma 4.1.** If $\bar{x}$ is a compliant vector and the response time of each $\tau_{j,\ell} \in H_c$ is at most $x_j + C_j$, then

$$W(t_b) \leq U(\tau)(y_{i,k} - t_b) + G(\bar{x}) + S(\bar{Y}).$$

**Proof.** We will say that a job $\tau_{j,\ell}$ is “executing at time instant $t_b – \delta$” if there is an $\varepsilon$ greater than 0 such that $\tau_{j,\ell}$ is executing over the entire interval $[t_b - \varepsilon, t_b)$. In Chapter 3, the presence of an idle CPU implied that at most $U^+ - 1$ tasks have work available for execution at time instant $t_b$, whereas here the same condition implies that at most $U^+ - 1$ jobs are available for execution. In Chapter 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 $\tau_i$ 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 categories of jobs that may contribute to $W(t_b)$: jobs that have s-release times before $t_b$ and are executing at time instant $t_b$ (Category 1) and jobs that have s-release times at or after $t_b$ (Category 2). Because there is an idle processor at time instant $t_b$, if a job has an s-release time before $t_b$ but is not executing at time instant $t_b$, then it has already completed by $t_b$ and does not contribute to $W(t_b)$.

**Category 1: Jobs With S-release Times Before $t_b$ Executing at $t_b$.** By the definition of $t_b$, there may be at most $U^+ - 1$ jobs executing at time instant $t_b$. We consider the jobs of each task $\tau_j$ that has jobs with s-release times before $t_b$ executing at time instant $t_b$. We will use $p$ (period) to index each executing job relative to the job with the most recent s-release within $\tau_j$: $p = 0$ 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 some $\tau_{j,\ell}$, then $\tau_{j,\ell}$ must complete by $x_j + C_j$ units after its s-release time, and must have an s-release time before $t_b - pT_j$. Therefore, $\tau_{j,\ell}$ must complete by time $t_b + x_j + C_j - pT_j$, and its contribution to $W_j(t_b)$ is at most $\min\{C_j, \max\{0, x_j + C_j - pT_j\}\}$

$$= g(\tau_j, x_j, p).$$

86
When \( p = 0 \) for some \( \tau_{j,\ell} \), \( x_j + C_j - pT_j \geq C_j \). Therefore \( g(\tau_j, x_j, p) = C_j \) by (4.1), so \( \tau_{j,\ell} \)'s contribution to \( W_j(t_b) \) is also at most \( g(\tau_j, x_j, p) \).

**Category 2: Jobs With S-release Times at or After \( t_b \).** We now consider jobs with s-release time at or after \( t_b \). By Lemma 3.2, each task \( \tau_j \) contributes at most \( U_j(y_{i,k} - t_b) + S_j \) units of work over \([t_b, y_{i,k})\). Cumulatively, all tasks contribute at most \( U(\tau)(y_{i,k} - t_b) + S(\bar{Y}) \) units of work over \([t_b, y_{i,k})\).

**Total Remaining Work.** \( W(t_b) \) contains at most \( U^+ - 1 \) jobs from Category 1, in addition to all jobs from Category 2, so \( W(t_b) \leq U(\tau)(y_{i,k} - t_b) + S(\bar{Y}) + G(\bar{x}). \)

We now provide a bound on \( W(r_{i,k}) \).

**Lemma 4.2.** If \( \bar{x} \) is a compliant vector and the response time of each \( \tau_{j,\ell} \in H_c \) is at most \( x_j + C_j \), then

\[
W(r_{i,k}) \leq U(\tau)Y_i + G(\bar{x}) + S(\bar{Y}).
\]

**Proof.** We have

\[
W(r_{i,k}) = \{ \text{Because at least } U^+ \text{ CPUs are busy in } [t_b, r_{i,k}) \}
\]
\[
W(t_b) - U^+ \cdot (r_{i,k} - t_b)
\]
\[
\leq \{ \text{Because } U^+ \geq U(\tau) \}
\]
\[
W(t_b) - U(\tau) \cdot (r_{i,k} - t_b)
\]
\[
\leq \{ \text{By Lemma 4.1} \}
\]
\[
U(\tau)(y_{i,k} - t_b) + G(\bar{x}) + S(\bar{Y}) - U(\tau) \cdot (r_{i,k} - t_b)
\]
\[
\leq \{ \text{Rewriting} \}
\]
\[
U(\tau)(y_{i,k} - r_{i,k}) + G(\bar{x}) + S(\bar{Y})
\]
\[
= \{ \text{By the definition of } Y_i \}
\]
\[
U(\tau)Y_i + G(\bar{x}) + S(\bar{Y}).
\]

We now use the previous lemma to bound the response time of a job under the same assumptions.
Lemma 4.3. If $\vec{x}$ is a compliant vector and the response time of each $\tau_{j,\ell} \in H_c$ is at most $x_j + C_j$, then the response time of $\tau_{i,k}$ is at most $x_i + C_i$.

Proof. After $r_{i,k}$, $\tau_{i,k}$ is continuously running until it is finished, except when all other CPUs are occupied by jobs from $H_c$. Recall that, by definition, $W(r_{i,k})$ is the total remaining work after time $r_{i,k}$ for jobs in $H$. We define $W_c(r_{i,k})$ as the total amount of remaining work after time $r_{i,k}$ for jobs in $H_c$. Because the upper bound in Lemma 4.2 assumes that all jobs (including $\tau_{i,k}$) run for their full worst-case execution times, Lemma 4.2 implies that

$$W_c(r_{i,k}) \leq U(\tau)Y_i + S(\vec{Y}) + G(\vec{x}) - C_i.$$ (4.5)

The total amount of time after $r_{i,k}$ during which $m$ CPUs are busy with work from $H_c$ can be at most

$$\frac{W_c(r_{i,k})}{m} \leq \frac{G(\vec{x}) + S(\vec{Y}) + U(\tau)Y_i - C_i}{m} \leq \frac{x_i}{m}.$$ (4.4)

Thus, $\tau_{i,k}$ is prevented from executing after its s-release time for at most $x_i$ time units, so its response time is at most $x_i + C_i$. □

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

**Theorem 4.1.** If $\vec{x}$ is a compliant vector, then each $\tau_{i,k}$ completes within $x_i + C_i$ units of its s-release time.

Proof. By inducting over the jobs of $H'$ using Lemma 4.3. □

### 4.3 The Minimum Compliant Vector

Theorem 4.1 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 \( \tau \) without implicit precedence constraints and corresponding assignment of \( \vec{Y} \), there exists a unique minimum compliant vector.

The analysis of the minimum compliant vector in Chapter 3 uses linear programming to compute and analyze the minimum compliant vector. Unfortunately, the definition of \( g(\tau_i, x_i, p) \) in (4.1) includes both a “min” and a “max”. This violates the convexity needed to use such LP techniques. Nonetheless, in Section 4.4, we do present a polynomial-time algorithm that can be used to compute the minimum compliant vector.

We first characterize the behavior of \( G(\vec{x}) \). We consider two vectors \( \vec{x} \) and \( \vec{z} \) that differ by a constant for some of their values, and are the same elsewhere. For example, \( \vec{x} = \langle 1, 2, 3 \rangle \) and \( \vec{z} = \langle 2, 2, 4 \rangle \) differ by exactly 1 in two places (the first and third) and are the same in the second; Lemma 4.4 would apply to \( \vec{x} \) and \( \vec{z} \) with \( k = 2 \).

**Lemma 4.4.** Suppose length-\( n \) vectors \( \vec{x} \) and \( \vec{z} \) differ at exactly \( k \) values, where \( k > 0 \), and for these values \( z_i = x_i + \delta \), where \( \delta \) is a positive constant. Denote \( w \triangleq \min \{ k, U^+ - 1 \} \).

The following inequality holds:

\[
G(\vec{x}) \leq G(\vec{z}) \leq G(\vec{x}) + \delta \cdot w. \tag{4.6}
\]

**Proof.** We will define a candidate sum for \( \vec{x} \) as any sum of \( U^+ - 1 \) distinct \( g(\tau_i, x_i, p) \) values as defined in (4.1). By (4.2), \( G(\vec{x}) \) is the largest candidate sum for \( \vec{x} \).

First, we prove \( G(\vec{x}) \leq G(\vec{z}) \). Consider the candidate sum \( N \) for \( \vec{z} \) computed by selecting the same \( i \) and \( p \) values as in \( G(\vec{x}) \). Because for all \( i, x_i \leq z_i \), \( G(\vec{x}) \leq N \). Because \( G(\vec{z}) \) must be the largest candidate sum for \( \vec{z} \), \( N \leq G(\vec{z}) \). Therefore, \( G(\vec{x}) \leq G(\vec{z}) \).

Next, we prove \( G(\vec{z}) \leq G(\vec{x}) + \delta \cdot w \) by contradiction. Suppose \( G(\vec{z}) > G(\vec{x}) + \delta \cdot w \). Consider the candidate sum \( T \) for \( \vec{x} \) computed by selecting the same \( i \) and \( p \) values as in \( G(\vec{z}) \). Observe that at most \( w \) terms contribute to the difference between \( G(\vec{z}) \) and \( T \). When two such terms differ, we have \( x_i = z_i - \delta \) (\( x_i = z_i \) otherwise). Thus, \( T \geq G(\vec{z}) - \delta \cdot w \), and hence, \( T > G(\vec{x}) \), which contradicts the fact that \( G(\vec{x}) \) is a maximal candidate sum for \( \vec{x} \). \( \square \)
We say that length- \( n \) \( \vec{x} \) is strictly smaller than length- \( n \) \( \vec{z} \) if for all \( i, x_i \leq z_i \) and there exists a \( j \) such that \( x_j < z_j \). Clearly \( \vec{z} \) cannot be considered “minimum” if there exists such an \( \vec{x} \). We next use Lemma 4.4 to characterize the minimum compliant vector.

**Lemma 4.5.** If \( \vec{z} \) is compliant and there is a \( j \) such that \( z_j > (G(\vec{z}) + S(\vec{Y}) + U(\tau)Y_j - C_j)/m \), then there exists a strictly smaller vector \( \vec{x} \) that is also compliant.

**Proof.** Define \( \vec{x} \) such that \( x_i = z_i \) for \( i \neq j \), and

\[
x_j = \frac{G(\vec{z}) + S(\vec{Y}) + U(\tau)Y_j - C_j}{m}.
\] (4.7)

In this case, \( \vec{x} \) and \( \vec{z} \) are of the form of Lemma 4.4 with \( k = 1 \). Therefore, \( G(\vec{x}) \leq G(\vec{z}) \).

We now have for all \( i \neq j \),

\[
\frac{G(\vec{x}) + S(\vec{Y}) + U(\tau)Y_i - C_i}{m} \leq \{ \text{Since } G(\vec{x}) \leq G(\vec{z}) \} \\
\frac{G(\vec{z}) + S(\vec{Y}) + U(\tau)Y_j - C_j}{m} \leq \{ \text{Since } \vec{z} \text{ is compliant, by (4.4)} \} \\
z_i = x_i.
\]

Also, by construction,

\[
\frac{G(\vec{x}) + S(\vec{Y}) + U(\tau)Y_j - C_j}{m} \leq \{ \text{Since } G(\vec{x}) \leq G(\vec{z}) \} \\
\frac{G(\vec{z}) + S(\vec{Y}) + U(\tau)Y_j - C_j}{m} = \{ \text{By (4.7)} \} \\
x_j.
\]
Therefore, $\bar{x}$ is compliant.

Lemma 4.5 demonstrates that each inequality in (4.4) should actually be an equality, or the vector cannot be the minimum. A minimum compliant vector must therefore be of the form

$$x_i = \frac{G(\bar{x}) + S(\bar{Y}) + U(\tau)Y_i - C_i}{m} \quad \forall i.$$  \hspace{1cm} (4.8)

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

$$s = \frac{G(\bar{x})}{m}  \hspace{1cm} (4.9)$$

such that

$$x_i = s + \frac{S(\bar{Y}) + U(\tau)Y_i - C_i}{m} \quad \forall i.$$  \hspace{1cm} (4.10)

We define some functions:

$$\bar{v}(s) \triangleq \bar{x} \text{ such that (4.10) holds},$$  \hspace{1cm} (4.11)

$$G(s) \triangleq G(\bar{v}(s)),$$  \hspace{1cm} (4.12)

$$M(s) \triangleq G(s) - ms.$$  \hspace{1cm} (4.13)

By (4.10), any minimum compliant vector must be $\bar{v}(s)$ for some $s$. Furthermore, $G(s)$ must equal $ms$, by (4.9). Therefore, $M(s) = 0$ if and only if $\bar{v}(s)$ is a compliant vector in the form of (4.8), and thus the minimum compliant vector. We are now ready to prove this section’s main result:

**Theorem 4.2.** For any given task set $\tau$, 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. The following lemma leads to the desired result as a corollary.

**Lemma 4.2.1.** $G(s)$ is continuous over the reals.
Proof. Let $\varepsilon > 0$ and $\delta_{\varepsilon} \triangleq \frac{\varepsilon}{U^+ - 1}$. Consider $s_0$ such that $|s - s_0| < \delta_{\varepsilon}$. Without loss of generality, assume $s < s_0$ (otherwise we can swap them.) Then $\vec{v}(s)$ and $\vec{v}(s_0)$ are of the form of $\vec{x}$ and $\vec{z}$, respectively, in Lemma 4.4, with $k = n$. Thus,

\[
G(\vec{v}(s)) \leq \{\text{By Lemma 4.4}\} \\
G(\vec{v}(s_0)) \\
\leq \{\text{By Lemma 4.4}\} \\
G(\vec{v}(s)) + \delta_{\varepsilon} \cdot (U^+ - 1) \\
= \{\text{By the definition of } \delta_{\varepsilon}\} \\
G(\vec{v}(s)) + \varepsilon.
\]

Therefore, $|G(s) - G(s_0)| \leq \varepsilon$, so $G(s)$ is continuous over the reals. \hfill \Box

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

**Lemma 4.2.2.** $M(0) > 0$

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

\[
M(0) \\
= \{\text{By (4.13) with } s = 0\} \\
G(0) \\
= \{\text{By (4.12) and (4.2)}\} \\
\sum_{U^+ - 1 \text{ largest}} g(\tau_i, v_i(0), p) \\
\geq \{\text{Since, by (4.1), } g(\tau_i, v_i(0), p) \text{ cannot be negative}\} \\
g(\tau_i, v_i(0), 0) \\
= \{\text{By the definition of } g(\tau_i, v_i(0), 0) \text{ in (4.1)}\} \\
\min \{C_i, \max \{0, v_i(0) + C_i\}\} \\
= \{\text{By (4.10) and (4.11), with } s = 0\}
\]

92
\[
\min \left\{ C_i, \max \left\{ 0, \frac{S(\bar{Y}) + U(\tau)Y_i - C_i}{m} + C_i \right\} \right\} = \{\text{Simplifying}\} \\
\min \left\{ C_i, \max \left\{ 0, \frac{S(\bar{Y}) + U(\tau)Y_i + (m-1)C_i}{m} \right\} \right\} > 0.
\]

\[\]

Lemma 4.2.3. \( M(C_{\max}) < 0. \)

Proof. By (4.1), \( g(\tau_i, v_i(C_{\max}), p) \leq C_i \) for any \( i \) and \( p \). Therefore, for any \( i \) and \( p \),

\[
g(\tau_i, v_i(C_{\max}), p) \leq C_{\max}. \tag{4.14}
\]

Therefore,

\[
M(C_{\max}) = \{\text{By (4.13) with } s = C_{\max}\} \\
G(C_{\max}) - mC_{\max} = \{\text{By (4.12)}\} \\
G(\bar{v}(C_{\max})) - mC_{\max} = \{\text{By (4.2)}\} \\
\sum_{U^+ - 1 \text{ largest}} g(\tau_i, v_i(C_{\max}), p) - mC_{\max} \leq \{\text{By (4.14)}\} \\
(U^+ - 1)C_{\max} - mC_{\max} \leq \{\text{Since } U^+ \leq m\} \\
-C_{\max} < 0.
\]
Lemma 4.2.4. There is an $s \in (0, C_{\text{max}})$ such that $M(s) = 0$.

Proof. By Lemma 4.2.1, Lemma 4.2.2, Lemma 4.2.3, and the Intermediate Value Theorem.

We now verify that the $s$ value of Lemma 4.2.4 is unique, using the following lemma.

Lemma 4.2.5. $s_1 \neq s_2$ implies $M(s_1) \neq M(s_2)$

Proof. Without loss of generality, assume $s_2 > s_1$ (otherwise, swap them). $\bar{v}(s_1)$ and $\bar{v}(s_2)$ are of the form of $\bar{x}$ and $\bar{z}$, respectively, with $\delta = (s_2 - s_1)$ and $k = n$, in Lemma 4.4. Therefore,

$$G(s_2) \leq G(s_1) + (s_2 - s_1)(U^+ - 1). \tag{4.15}$$

Thus,

$$M(s_2) - M(s_1)$$

$$= \{\text{By (4.13)}\}$$

$$G(s_2) - ms_2 - G(s_1) + ms_1$$

$$\leq \{\text{By (4.15)}\}$$

$$G(s_1) + (s_2 - s_1)(U^+ - 1) - ms_2 - G(s_1) + ms_1$$

$$= \{\text{Simplifying}\}$$

$$(s_2 - s_1)(U^+ - 1 - m)$$

$$\leq \{\text{Since } U^+ \leq m\}$$

$$-1(s_2 - s_1)$$

$$< 0.$$

Therefore, $M(s_1) \neq M(s_2)$.

Lemma 4.2.5 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 Lemma 4.2.4 is unique.
<table>
<thead>
<tr>
<th>$\tau_1$</th>
<th>$\tau_2$</th>
<th>$\tau_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>12</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>$Y_i$</td>
<td>$Y_i$</td>
<td>$Y_i$</td>
</tr>
</tbody>
</table>

Figure 4.2: Two-CPU task system example for Section 4.4

In Lemma 4.2.4 also leads to Theorem 4.3, which provides a response-time bound that can be quickly calculated.

**Theorem 4.3.** The response time of any job of any task $\tau_i$ cannot exceed $C_{\text{max}} + \frac{S(\bar{Y}) + U(\tau)Y_i - C_i}{m} + C_i$.

**Proof.** Follows from Lemma 4.2.4, (4.10), and Theorem 4.1. \hfill \square

### 4.4 Computation Algorithm

We now show how to compute the minimum compliant vector for a task system $\tau$ in time polynomial with respect to the size of $\tau$ and the number of processors. $G(s)$ as defined in (4.12) is a piecewise linear function; our algorithm works by tracing $G(s)$ until we find a fixed point $G(s) = ms$.

In order to assist the reader’s understanding of this algorithm, we provide an example task system in Figure 4.2.\textsuperscript{2} Simple calculations reveal that, for this system, $S(\bar{Y}) = 0$ and $U(\tau) = 2$. Furthermore, in a two-CPU system, by Definition 4.1, we only need to consider $p = 0$. A graph of the relevant $g(\tau_i, v_i(s), 0)$ functions with respect to $s$ is provided in Figure 4.3.

We define the slope at point $s$ of a piecewise linear function $f(s)$ to be $\lim_{\varepsilon \to 0^+} \frac{f(s + \varepsilon) - f(s)}{\varepsilon}$. This definition differs from the common notion of derivative in that its limit is taken from the right; it is thus defined for all real $s$. For example, $g(\tau_1, v_1(s), 0)$ in Figure 4.3 has a slope of 1 at $s = -22$, but is not differentiable at $s = -22$.

For each value of $s$ we will define $g(\tau_i, v_i(s), p)$ as being in one of three states, depending on the value of $v_i(s) + C_i - pT_i$:

- If $v_i(s) + C_i - pT_i < 0$, then $g(\tau_i, v_i(s), p)$ is in state 0, is equal to 0, and has a slope of 0.
- $g(\tau_1, v_1(s), 0)$ in Figure 4.3 is in state 0 in the interval $(-\infty, -22)$.

\textsuperscript{2}In this system, the worst-case execution time of $\tau_2$ exceeds its deadline, so it appears that it is impossible for $\tau_2$ to meet its deadline. However, because execution times given are worst-case rather than exact, it is actually possible for this job to complete before its deadline. Furthermore, here we are interested in response-time bounds rather than hard deadlines.
• If $0 \leq v_i(s) + C_i - p T_i < C_i$, then $g(\tau_i, v_i(s), p)$ is in state 1, is equal to $v_i(s) + C_i - p T_i$, and has a slope of 1. $g(\tau_1, v_1(s), 0)$ in Figure 4.3 is in state 1 in the interval $[-22, -18)$.

• If $C_i \leq v_i(s) + C_i - p T_i$, then $g(\tau_i, v_i(s), p)$ is in state 2, is equal to $C_i$, and has a slope of 0. $g(\tau_1, v_1(s), 0)$ in Figure 4.3 is in state 2 in the interval $[-18, \infty)$.

In order to analyze the piecewise linear function $G(s)$, we will need to determine where the slope changes. To do so, we need to determine which $g(\tau_i, v_i(s), p)$ components contribute to $G(s)$ for various intervals. For some intervals, the choice is arbitrary. For example, the task system in Figure 4.3 has only one $g(\tau_i, v_i(s), p)$ component contributing to $G(s)$, because $m - 1 = 2 - 1 = 1$. However, for $s < -22$ all $g(\tau_i, v_i(s), p)$ components equal zero. We provide a sufficient solution by arbitrarily tracking some valid set of $g(\tau_i, v_i(s), p)$ components.

We will create a set points of tuples, one for each possible change in the slope of $G(s)$. (Each will have an associated $s$ value, but there could be multiple possible changes at the same $s$ value.) Each tuple will identify a point where some $g(\tau_{i_0}, v_{i_0}(s), p_0)$ in state $h_0$ is replaced by some $g(\tau_{i_1}, v_{i_1}(s), p_1)$ in state $h_1$. Such a tuple will be of the form $(s, i_0, p_0, h_0, i_1, p_1, h_1)$. In some cases, more than one old component may be appropriate. To handle these cases efficiently, any of $i_0, p_0,$ or $h_0$ may be set to $\ast$, which is defined as matching any value of the relevant parameter. For example, the tuple $(s, \ast, \ast, 0, i_1, p_1, 1)$ indicates that any arbitrary $g(\tau_{i_0}, v_{i_0}(s), p_0)$ in state 0 should be replaced by $g(\tau_{i_1}, v_{i_1}(s), p_1)$ in state 1.

The slope of $G(s)$ may change in any of the following cases:
1. Some \( g(\tau_i, v_i(s), p) \) changes from state 0 to state 1. This occurs where \( v_i(s) + C_i - pT_i = 0 \). The resulting tuple will be \((s, *, *, 0, i, p, 1)\), as we can view \( g(\tau_i, v_i(s), p_i) \) as replacing any \( g(\tau_j, v_j(s), p_j) \) in state 0 in the system—they all have value 0. This change occurs exactly once per \( g(\tau_i, v_i(s), p) \) and therefore \( m - 1 \) times per task (once per value of \( p \)), for a total of \( O(mn) \) times for the system. In Figure 4.3, this state change occurs for \( g(\tau_1, v_1(s), 0) \) at \( s = -22 \), for \( g(\tau_2, v_2(s), 0) \) at \( s = -13 \), and for \( g(\tau_3, v_3(s), 0) \) at \( s = -16 \).

2. Some \( g(\tau_i, v_i(s), p) \) changes from state 1 to state 2. This occurs where \( v_i(s) + C_i - pT_i = C_i \) (so \( v_i(s) = pT_i \)). The resulting tuple will be \((s, i, p, 1, i, p, 2)\). As above, this change occurs \( O(mn) \) times for the system. In Figure 4.3, this state change occurs for \( g(\tau_1, v_1(s), 0) \) at \( s = -18 \), for \( g(\tau_2, v_2(s), 0) \) at \( s = -7 \), and for \( g(\tau_3, v_3(s), 0) \) at \( s = -4 \).

3. Some \( g(\tau_i, v_i(s), p_i) \) is in state 1 and crosses \( C_j \), and thus potentially crosses \( g(\tau_j, v_j(s), p_j) \) (for some \( p_j \)) where the latter is in state 2. This occurs when \( C_i > C_j \) and \( v_i(s) + C_i - p_iT_i = C_j \). The resulting tuple will be \((s, j, *, 2, i, p, 1)\). This point may exist at most \( n - 1 \) times per \( g(\tau_i, v_i(s), p) \) (in the worst case, \( g(\tau_i, v_i(s), p) \) crosses one \( g(\tau_j, v_j(s), p_j) \) for each other \( \tau_j \)), so occurs at most \( O(mn^2) \) times for the system. In Figure 4.3, this point does not occur for \( \tau_1 \) (as \( C_1 \) is the smallest value in the system), occurs for \( g(\tau_2, v_2(s), 0) \) with \( \tau_1 \) at \( s = -9 \), and occurs for \( g(\tau_3, v_3(s), 0) \) with \( \tau_1 \) at \( s = -12 \) and with \( \tau_2 \) at \( s = -10 \). (Although \( g(\tau_3, v_3(s), 0) \) does not actually cross \( g(\tau_2, v_2(s), 0) \) at \( s = -10 \), our algorithm nonetheless records the point where \( g(\tau_3, v_3(s), 0) \) crosses \( C_2 \).)

In order to track \( G(s) \), we order the tuples in points by \( s \) value, breaking ties in favor of tuples indicating a change in state for a particular \( g(\tau_i, v_i(s), p) \) component. We create a list active containing tuples \((i, p, h)\), each representing the corresponding \( g(\tau_i, v_i(s), p) \) in state \( h \) that contributes its value to \( G(s) \). For \( s \) smaller than the smallest in points, we may arbitrarily make \( U^+ - 1 \) choices of \( g(\tau_i, v_i(s), p) \) components, each in state 0. Therefore, we initialize active to an arbitrary choice of \( U^+ - 1 \) tuples of the form \((i, p, 0)\).

The appropriate \( s \) value is computed using Algorithm 4.1, which works by tracing the piecewise linear function and checking for \( G(s) = ms \) (as per (4.9), (4.11), and (4.12)) in each segment.

As an example, suppose active is initialized to \( \{(3, 0, 0)\} \), which represents \( g(\tau_3, v_3(s), 0) \) in state 0. The first tuple in points is \(( -22, *, 0, 0, 1, 0, 1) \), representing the leftmost slope change in
The new state value of $s$ simply requires sorting. The complexity of Algorithm 4.1 is $O(n^2)$.

Once the next iteration the correct value of $G$ is assigned the value is of 1. We now know the slope $m \cdot s$ would hold, so checking for matches will require $O(mn^2)$ operations over the execution of the algorithm. Each match requires $O(1)$ time to process, so the complexity of Algorithm 4.1 is $O(mn^2)$. Computing points requires $O(mn^2)$ time, and sorting requires $O(mn^2 \log(mn))$ time, so the complexity of computing $s$ is $O(mn^2 \log(mn) + m^2 n^2)$. Once an $s$ value has been computed using Algorithm 4.1, the correct minimum compliant vector is simply $\vec{\gamma}(s)$, which can be computed in $O(n)$ time.

Algorithm 4.1: Computation of minimum compliant vector.

```plaintext
slope := 0;
current := 0;
foreach $(s_1, i_1, p_1, h_1, i_2, p_2, h_2) \in$ points do
  if $(i_1, p_1, h_1)$ matches some $(i, p, h)$ in active then
    Replace $(i, p, h)$ in active with $(i_2, p_2, h_2);
    if $h_2 = 1$ then
      // Changing to state 1 means slope increases.
      slope := slope + 1;
    else
      // Must be changing away from state 1 or $(s_1, i_1, p_1, h_1, i_2, p_2, h_2)$
      would’t be in points
      slope := slope - 1;
    $s_2$ := next $s$ value from points, or $C_{\max}$ if there is no such value;
    $s$ := $\frac{current - slope \cdot s_1}{m \cdot slope}$;
    if $s \in [s_1, s_2)$ then
      return $s$;
    current := current + slope \cdot (s_2 - s_1);
  if
```

Figure 4.3. This tuple will match the single tuple in active, so active will become $(1, 0, 1)$. slope is used to track the slope between $s_1$ and the next $s$ value in points (which is called $s_2$). current is used to represent the correct value of $G(s_1)$. In this case, the current interval of interest is $-22 \leq s < -18$. The new state $h_2$ is 1, so the slope (which was initially 0) will be incremented by 1, resulting in a new slope of 1. We now know the slope slope $= 1$ of $G(s)$ over $[-22, -18]$ and its value $G(s_1) = current = 0$ at $s_1 = -22$. We therefore compute the point where $G(s) = ms$ would hold, assuming a linear function that is equal to the correct piecewise linear function over the interval of interest. In this case, $s$ is assigned the value $\frac{0 - (-22)}{2 - 1} = 22$, which is not in $[-22, -18]$, so the desired value of $s$ for the algorithm is not in the current interval of interest. We do not return, so we update the value current to match the value of $G(s_2)$ at the end of the current interval of interest (and thus in the next iteration the correct value of $G(s_1)$). In this case, current will be assigned to $0 + 1 \cdot 4 = 4$.

The set points is of size $O(mn^2)$, and the set active is of size $O(m)$, so checking for matches will require $O(mn^2)$ operations over the execution of the algorithm. Each match requires $O(1)$ time to process, so the complexity of Algorithm 4.1 is $O(mn^2)$. Computing points requires $O(mn^2)$ time, and sorting requires $O(mn^2 \log(mn))$ time, so the complexity of computing $s$ is $O(mn^2 \log(mn) + m^2 n^2)$. Once an $s$ value has been computed using Algorithm 4.1, the correct minimum compliant vector is simply $\vec{\gamma}(s)$, which can be computed in $O(n)$ time.
4.5 Evaluation

As discussed in Section 4.3, we do not have an LP model for this type of system. Therefore, we do not have an efficient means of calculating the “best” choice of $\vec{Y}$. Therefore, to provide a comparison between systems with and without intra-task precedence constraints, we consider scheduling under G-EDF with implicit deadlines.

Under implicit deadlines, each $Y_i = T_i$. Therefore, for arbitrary $\tau_i$,

$$S_i(Y_i) = \{ \text{By the definition of } S_i(Y_i) \text{ in (3.8)} \}$$

$$C_i \cdot \max \left\{ 0, 1 - \frac{Y_i}{T_i} \right\}$$

$$= \{ \text{Because } Y_i = T_i \}$$

$$= 0.$$

Thus, by the definition of $S(\vec{Y})$ in (4.3), $S(\vec{Y}) = 0$. Combining this result with Theorem 4.3 and the necessary condition that $U(\tau) \leq m$, the response time of any job of any task $\tau_i$ must be upper-bounded by $C_{\text{max}} + Y_i + \frac{m-1}{m}C_i$. Therefore, the lateness of any job of $\tau_i$ must be no greater than $C_{\text{max}} + \frac{m-1}{m}C_i$.

In order to evaluate the improvement to the bounds we obtain by eliminating implicit precedence constraints, we compared our results to those available using the EDF-CVA2 technique in Chapter 3. As shown in Chapter 3, this technique typically provides the best bounds for highly utilized systems scheduled using implicit-deadline G-EDF. (As discussed in Chapter 3, EDF-CVA can also analyze the same systems, but typically provides inferior bounds to EDF-CVA2 for highly utilized systems.)

Our experiments were intended to show how varying different task system parameters affected the improvements available by removing intra-task precedence constraints. All experiments were done with processor counts of 4, 8, and 16. We used uniform distributions for the task worst-case execution times and utilizations, described below. We determined the effects of varying each of four parameters: mean worst-case execution time ($\bar{C}$), standard deviation of worst-case execution time ($C_\sigma$), mean utilization ($\bar{U}$), and standard deviation of utilization ($U_\sigma$). We performed one experiment for each parameter. For mean $x$ and standard deviation $\sigma$, values were chosen uniformly over $(x - \sigma\sqrt{3}, x + \sigma\sqrt{3})$. 

99
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 1,000 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 lateness bound under EDF-CVA2, \( \delta \), and using Theorem 4.1, \( \delta' \). For each set of 1,000 task sets we computed \( \bar{\delta} \) (the mean value of \( \delta \)) and \( \bar{\delta}' \) (the mean value of \( \delta' \)). The \textit{absolute improvement} for each set of sets is defined as \( \bar{\delta} - \bar{\delta}' \), and the \textit{relative improvement} for each set of sets is defined as \( (\bar{\delta} - \bar{\delta}')/\bar{\delta} \).

Results of the two experiments varying WCET parameters are presented in Figure 4.4. In Figure 4.4(a), we use \( C_\sigma = 5.8, \bar{U} = 0.5, \) and \( U_\sigma = 0.29 \), and we depict the relative improvement with respect to \( \bar{C} \). For larger \( \bar{C} \), the relative improvement increases, but the dependence on \( \bar{C} \) is small. This means that EDF-CVA2 is slightly more sensitive to \( \bar{C} \) than the techniques proposed in this chapter. In Figure 4.4(b), we use \( \bar{C} = 180, \bar{U} = 0.5, \) and \( U_\sigma = 0.29 \), and we depict the relative improvement with respect to \( C_\sigma \). Here we see a decreasing trend. When \( \bar{C} \) is held constant but \( C_\sigma \) is increased, the largest few WCET values tend to increase. We see that the techniques proposed in this chapter are slightly more sensitive to this effect than EDF-CVA2.

Results of the two experiments varying utilization parameters are presented in Figure 4.5. In Figure 4.5(a), we use \( \bar{C} = 10, C_\sigma = 2.9, \) and \( U_\sigma = 0.029 \), and we depict the relative improvement with respect to \( \bar{U} \). We see here that the relative improvement increases dramatically as mean utilizations are increased, indicating that EDF-CVA2 is significantly more sensitive to \( \bar{U} \). This effect is not surprising, because no \( U_i \) term appears in Definition 4.1. In Figure 4.5(b), we use \( \bar{C} = 10, C_\sigma = 2.9, \) and \( \bar{U} = 0.5 \), and we depict the relative improvement with respect to \( U_\sigma \). In an analogous fashion to the WCET experiments, increasing \( U_\sigma \) while holding \( \bar{U} \) constant resulted in increasing the largest few utilizations. In this case, EDF-CVA2 is sensitive to this effect, but the techniques proposed in this chapter do not depend on utilizations. Thus, the relative improvement grows with increasing \( U_\sigma \).

Overall, we see that the relative improvement 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 EDF-CVA2 bounds 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.

### 4.6 Conclusion

GEL 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 GEL scheduling may be even more useful for SRT workloads in which jobs may be released as separate threads that can
Figure 4.5: Results of experiments varying utilization parameters.

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.