Soft Real-Time Gang Scheduling

Shareef Ahmed and James H. Anderson
Department of Computer Science, University of North Carolina at Chapel Hill
{shareef, anderson}@cs.unc.edu

***Abstract*—Due to the emergence of parallel architectures and parallel programming frameworks, modern real-time applications are often composed of parallel tasks that can occupy multiple processors at the same time. Among parallel task models, *gang scheduling* has received much attention in recent years due to its performance efficiency and applicability to parallel architectures such as graphics processing units. Despite this attention, the soft real-time (SRT) scheduling of gang tasks has received little attention. This paper, for the first time, considers the SRT-feasibility problem for gang tasks. Necessary and sufficient feasibility conditions are presented that relate the SRT-feasibility problem to the HRT-feasibility problem of "equivalent" task systems. Based on these conditions, intractability results for SRT gang scheduling are derived. This paper also presents server-based scheduling policies, corresponding schedulability tests, and an improved schedulability condition for the global-earliest-deadline-first (GEDF) scheduling of gang tasks. Moreover, GEDF is shown to be non-optimal in scheduling SRT gang tasks.**

I. Introduction

Recent advances in multicore platforms, hardware accelerators (*e.g.*, graphics processing units (GPUs)), and parallel-programming models (*e.g.*, OpenMP) have led to the widespread adoption of parallelization in supporting real-time workloads. These advances have contributed to the growth of artificial intelligence (AI)-based autonomous applications, which heavily rely on the ability to group multiple application threads into gangs for efficient computation. To support such applications in safety-critical real-time systems, analyzable scheduling techniques are needed for *gang-based* task models where k parallel threads execute in unison on k processors. Due to its significance, gang scheduling has garnered much attention in recent years [6], [8], [10], [12], [16], [21].

With the exception of [8], all prior work on gang scheduling pertains to hard real-time (HRT) systems, where each instance of a task must meet its deadline. In contrast, for soft real-time (SRT) systems, a task instance can be *tardy* and may only require bounded tardiness by completing execution within a bounded amount of time after its deadline. The lone work on SRT gang scheduling cited above, due to Dong *et al.* [8], gives a sufficient condition for bounded tardiness and a tardiness bound for gang tasks under global-earliest-deadline-first (GEDF) scheduling.

Complexities in SRT gang scheduling. Despite its importance, the work by Dong *et al.* illustrates the pessimism inherent in SRT gang scheduling. Since the seminal work of Devi and Anderson [5], it has been known that GEDF can ensure bounded tardiness for any ordinary sporadic task system without causing any utilization loss. Unfortunately, this is not the case for gang tasks, as there exist gang task systems with utilizations greater than but arbitrarily close to 1.0 for which tardiness is unbounded [8].

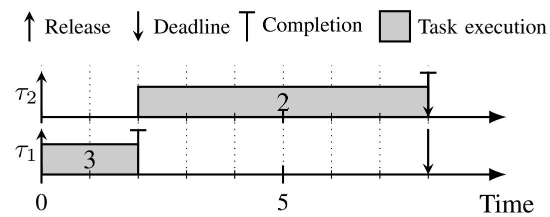


Fig. 1: Two gang tasks on four processors. The number inside each execution block denotes the degree of parallelism.

Additionally, existing schedulability conditions for SRT gang tasks can be overly pessimistic. For example, consider two gang tasks τ_1 and τ_2 with a degree of parallelism (*i.e.*, the number of processors required to execute the task) of three and two, respectively. Task τ_1 (resp., τ_2) has a worst-case execution time of 2.0 (resp., 6.0) time units, while each has a period and relative deadline of 8.0 time units. If scheduled on a four-core platform by GEDF, the schedulability condition by Dong *et al.* [8] cannot guarantee bounded tardiness for this task system (we elaborate later in Sec. V). However, as seen in Fig. 1, this task system is actually HRT-schedulable under GEDF (this is true for any release pattern of these tasks).

Motivated by this, we consider herein the SRT-feasibility problem for preemptive gang scheduling, which asks whether a given gang task system can be scheduled such that each instance of a task has bounded tardiness. We give necessary and sufficient conditions for SRT feasibility, based on which we show that the SRT-feasibility problem is NP-hard. To the best of our knowledge, this is the first intractability result regarding SRT scheduling where only bounded tardiness is required. Furthermore, we give server-based scheduling policies for gang tasks and corresponding schedulability conditions for bounded tardiness. We also show that GEDF is non-optimal in scheduling SRT gang tasks and give an improved condition for achieving bounded tardiness under GEDF.

Related work. Most prior work on gang scheduling focuses on preemptive gang scheduling for HRT systems. It is known that optimally scheduling HRT gang tasks is NP-hard in the strong sense even when all tasks have the same period and relative deadline [14]. Schedulability tests for HRT preemptive

Work was supported by NSF grants CPS 1837337, CPS 2038855, CPS 2038960, and CNS 2151829, ARO grant W911NF-20-1-0237, and ONR grant N00014-20-1-2698.

gang scheduling have been presented for GEDF [6], [12], [16], [21], fixed-priority (FP) scheduling [10], [16], and stationary scheduling algorithms [24]. It has also been shown that schedules for periodic HRT gang tasks can be optimally constructed (offline) in polynomial time for a fixed number of processors [11]. Recently, more expressive gang task models have been considered, such as gang tasks with precedence constraints [3] and mixed-criticality gang tasks [4]. For the non-preemptive scheduling of HRT gang tasks, schedulability tests have been presented for GEDF and FP scheduling [7], [15], [20]. For SRT scheduling, both a schedulability test and tardiness bound under GEDF have been presented [8].

SRT scheduling of ordinary sporadic tasks and directed-acyclic-graph-(DAG)-based tasks has been well studied. A wide class of schedulers including GEDF and *first-in-first-out* (FIFO) schedulers can ensure bounded response times for ordinary sporadic tasks and DAG-based tasks on identical multiprocessor platforms without incurring any utilization loss [1], [2], [5], [9], [17]–[19]. Recent work has shown that, under GEDF and its variants, such a property holds even when execution speeds of different processors vary, or each task can only be scheduled on certain processors [22], [23], [25].

Contributions. Our contribution is fivefold:

First, we develop necessary and sufficient conditions for the SRT-feasibility problem. Each condition involves associating a SRT task system with a corresponding HRT one that is "equivalent" in a feasibility sense.

Second, by utilizing these feasiblity conditions, we show that the SRT-feasibility problem for gang tasks is NP-hard.

Third, leveraging the sufficient condition for SRT-feasibility, we propose server-based scheduling policies and corresponding schedulability tests for gang tasks.

Fourth, we demonstrate that GEDF scheduling is not optimal for scheduling SRT gang tasks. Furthermore, we present an improved schedulability test for gang scheduling under GEDF that outperforms existing tests (although it may result in a worse tardiness bound).

Finally, we present an experimental evaluation of our results that illustrates their benefits.

Organization. After covering needed background (Sec. II), we discuss the SRT-feasibility of gang tasks (Sec. III), provide our server-based scheduling policies (Sec. IV), discuss the GEDF scheduling of gang tasks (Sec. V), present our experiments (Sec. VI), and conclude (Sec. VII).

II. Preliminaries

We consider a set Γ of n sporadic gang tasks to be globally scheduled on M identical processors. Each gang task τ_i releases a potentially infinite sequence of jobs $\tau_{i,1}, \tau_{i,2}, \ldots$. Each sporadic (resp., periodic) gang task τ_i has a *period* T_i, which is the minimum (resp., exact) separation time between any two consecutive job releases of τ_i. The *relative deadline* of τ_i is denoted by D_i. We consider implicit deadlines, meaning that $D_i = T_i$ holds for each τ_i. Task τ_i has a *worst-case execution time* (WCET) of C_i. The execution time of job $\tau_{i,j}$ is $C_{i,j}$. Each task τ_i has a *degree of parallelism* m_i, which is the number of simultaneously available processors required to execute any job of τ_i. Thus, the *worst-case execution requirement* (WCER) of each job of τ_i can be represented by a rectangle of area $m_i \times C_i$ in a schedule. We let $C_{max} = \max_i\{C_i\}$, $C_{min} = \min_i\{C_i\}$, and $T_{max} = \max_i\{T_i\}$. Jobs of τ_i are sequential, meaning that no two jobs of τ_i can execute in parallel.

TABLE I: Notation summary.

Symbol	Meaning	Symbol	Meaning
n	No. of gang tasks	m_i	Degree of parallelism of τ_i
M	No. of processors	λ_i	Horizontal utilization of τ_i
Γ	Task system	$r_{i,j}$	Release time of $\tau_{i,j}$
τ_i	i^{th} task of Γ	$d_{i,j}$	Deadline of $\tau_{i,j}$
$\tau_{i,j}$	j^{th} job of τ_i	$f_{i,j}$	Finish time of $\tau_{i,j}$
T_i	Period of τ_i	H	Hyperperiod
C_i	WCET of τ_i	h_i	H/T_i
D_i	Rel. deadline of τ_i	Γ_s^H	Set of servers
u_i	Utilization of τ_i	$\mathcal{I}$	Ideal schedule of Γ
U	$\sum_i u_i$	$\mathcal{S}$	A schedule of Γ
C_{max}	$\max_i C_i$	lag	lag of a task or a job
T_{max}	$\max_i T_i$	LAG	LAG of Γ or a set of jobs

The *release time*, *deadline*, and *finish time* of $\tau_{i,j}$ are denoted by $r_{i,j}$, $d_{i,j}$, and $f_{i,j}$, respectively. The *response time* and *tardiness* of $\tau_{i,j}$ are $f_{i,j} - r_{i,j}$ and $\max\{0, f_{i,j} - d_{i,j}\}$, respectively. Task τ_i's response time (resp., tardiness) is the maximum response time (resp., tardiness) of any of its jobs.

The *utilization* of τ_i is $u_i = (C_i \times m_i)/T_i$. Note that, unlike sporadic tasks, u_i can exceed 1.0 for a gang task τ_i. The *total utilization* of task system Γ is $U = \sum_{i=1}^{n} u_i$. The *horizontal utilization* λ_i of τ_i is C_i/T_i. The *hyperperiod* H is the least common multiple of all periods. We let h_i denote H/T_i.

A periodic gang task τ_i has an offset ϕ_i that denotes the release time of the first job of τ_i. For brevity, we denote a periodic (resp., sporadic) gang task by (ϕ_i, T_i, C_i, m_i) (resp., (T_i, C_i, m_i)). We summarize all introduced notation in Tbl. I.

We assume time to be discrete and a unit of time to be 1.0. All scheduling decisions and job releases occur at integer points in time. We also assume all task parameters to be integers. Therefore, when a job $\tau_{i,j}$ executes during a unit interval $[t-1, t)$, it continuously executes during $[t-1, t)$.

Def. 1. *A job $\tau_{i,j}$ of task $\tau_i \in \Gamma$ is* pending *at time t if $r_{i,j} \leq t < f_{i,j}$ holds. $\tau_{i,j}$ is* ready *at time t if it is pending at time t and job $\tau_{i,j-1}$ (if $j > 1$) finishes execution by time t.*

Concrete and non-concrete tasks system. A task system is *concrete* if the release time and actual execution time of every job of each task is known, and *non-concrete*, otherwise. Infinitely many concrete task systems can be specified for a non-concrete task system and we call each such a concrete task system a *concrete instantiation* of the non-concrete system.

Feasibility and schedulability. A task system Γ is *SRT-schedulable* (resp., *HRT-schedulable*) under a scheduling algorithm $\mathcal{A}$ if and only if tardiness of each task of Γ is bounded (resp., 0) under $\mathcal{A}$ for any concrete instantiation of Γ. Task system Γ is *SRT-feasible* (resp., *HRT-feasible*) if and only if Γ is SRT-schedulable (resp., HRT-schedulable) under some scheduling algorithm. A scheduling algorithm is *SRT-optimal* (resp., *HRT-optimal*) if and only if it can schedule any SRT-feasible (resp., HRT-feasible) task system.

Parallelism-induced idleness. When scheduling gang tasks, *parallelism-induced idleness* may occur [8]. A time instant t is parallelism-induced idle if there is an idle processor at time t and a job $\tau_{i,j}$ is pending but unscheduled at time t due to the lack of m_i available processors. In Fig. 1, there is an idle processor during time interval $[0, 2)$. Although τ_2 has a pending job during this interval, it cannot execute, as the number of available processor is less than m_2. Thus, there is parallelism-induced idleness during $[0, 2)$.

III. SRT-Feasibility of Gang Tasks

In this section, we consider the problem of determining the SRT-feasibility of a gang task system. In this section, we assume the following, which we justify in Lemma 2 in the context of SRT-feasibility.

A Each job of any task τ_i executes for its WCET C_i.

Lemma 1. *If a concrete instantiation Γ_c of a non-concrete task system Γ, satisfying Asm. A, is SRT-schedulable by some algorithm, then any concrete instantiation Γ'_c of Γ that differs from Γ_c only in job execution times is also SRT-schedulable by some algorithm.*

Proof. Let $\mathcal{S}$ be a schedule of Γ_c such that each task has bounded tardiness. Using $\mathcal{S}$, we construct a schedule $\mathcal{S}'$ of Γ'_c. In $\mathcal{S}'$, each job $\tau_{i,j}$ executes whenever it is scheduled in $\mathcal{S}$ until it completes. If $\tau_{i,j}$ finishes at time t in $\mathcal{S}$ and at time $t' < t$ in $\mathcal{S}'$, then at any time $t'' \in [t', t)$ when $\tau_{i,j}$ is scheduled in $\mathcal{S}$, $\mathcal{S}'$ keeps the processors $\tau_{i,j}$ occupies in $\mathcal{S}$ idle. Thus, each job has bounded tardiness, as it finishes no later in $\mathcal{S}'$ than $\mathcal{S}$. □

Lemma 2. *If every concrete instantiation of Γ satisfying Asm. A is SRT-schedulable by some algorithm, then Γ is SRT-feasible.*

Proof. Assume that Γ is not SRT-feasible. Then, there exists a concrete instantiation Γ'_c of Γ that is not SRT-schedulable by any algorithm. By the assumption of the lemma, Γ'_c does not satisfy Asm. A. Let Γ_c be the concrete instantiation of Γ such that Γ_c satisfies Asm. A, and it only differs from Γ'_c in terms of job execution times. Since Γ_c satisfies Asm. A, it is SRT-schedulable by some algorithm. Thus, by Lemma 1, Γ'_c is SRT-schedulable by some algorithm, a contradiction. □

Before proving the hardness of the SRT-feasibility problem for gang tasks, we first give a necessary condition (Sec. III-A) and a sufficient condition (Sec. III-B) for bounded tardiness.

A. Necessary Condition for SRT-Feasibility

We give a necessary condition for SRT-feasibility in Lemma 3, which utilizes the following definition.

Def. 2. *Given the sporadic task system $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$, let $\Gamma^{kH} = \{\tau_1^{kH}, \tau_2^{kH}, \ldots, \tau_n^{kH}\}$ be a set of implicit-deadline periodic gang tasks such that k is a positive integer and $\tau_i^{kH} = (0, kH, kh_iC_i, m_i)$.*

Lemma 3. *If Γ is SRT-feasible, then there is a positive integer k such that the periodic task system Γ^{kH} is HRT-feasible.*

Proof. Let Γ_c be a concrete instantiation of Γ such that each task periodically releases its jobs starting from time 0. Since Γ is SRT-feasible, there is a schedule $\mathcal{S}$ of Γ_c such that each task τ_i has bounded tardiness. Let x_i be τ_i's tardiness in $\mathcal{S}$. Let $e_i(t) \geq 0$ be the remaining execution time of all jobs of τ_i released *before* time t in $\mathcal{S}$.

We now consider the $e_i(t)$ values at times $0, H, 2H, \ldots$. In Γ_c, each task τ_i releases a job at time $t \in \{0, H, 2H, \ldots\}$. By the definition of $e_i(t)$, at any time $t \in \{0, H, 2H, \ldots\}$, $e_i(t)$ does not include the execution time of τ_i's job released at time t. Since τ_i's tardiness is x_i, at any time $t \in \{0, H, 2H, \ldots\}$, τ_i's jobs released before time t have at most x_i time units of execution remaining at time t. (Note that τ_i's jobs must execute in sequence, so if its tardiness is x_i, then all of its tardy jobs at a hyperperiod boundary must be complete within x_i time units beyond that boundary.) Thus, at any time $t \in \{0, H, 2H, \ldots\}$, $0 \leq e_i(t) \leq x_i$ holds for all i.

Therefore, since $e_i(t)$ is an integer, $e_i(t)$ can take at most $x_i + 1$ distinct values at any time $t \in \{0, H, 2H, \ldots\}$. Thus, the tuple $(e_1(t), e_2(t), \ldots, e_n(t))$ takes on one of $X = \prod_{i=1}^{n}(x_i + 1)$ distinct values at any time $t \in \{0, H, 2H, \ldots\}$. Therefore, there must be a pair of integers $b < d \leq X + 1$ such that $(e_1(bH), e_2(bH), \ldots, e_n(bH)) = (e_1(dH), e_2(dH), \ldots, e_n(dH))$ holds. Let $k = d - b$.

Since τ_i releases jobs periodically in Γ_c, it releases $(d - b) \cdot \frac{H}{T_i} = kh_i$ jobs in $[bH, dH)$. Thus, by Asm. A, during $[bH, dH)$, τ_i executes for $e_i(bH) + kh_iC_i - e_i(dH) = kh_iC_i$ time units. Let $\mathcal{S}_k$ be the portion of schedule $\mathcal{S}$ during $[bH, dH)$. Using $\mathcal{S}_k$, we can create a HRT-feasible schedule $\mathcal{S}^{kH}$ of Γ^{kH}. $\mathcal{S}^{kH}$ mimics $\mathcal{S}_k$ with the exception that τ_i^{kH} executes in $\mathcal{S}^{kH}$ instead of τ_i whenever τ_i is scheduled in $\mathcal{S}_k$. Since every task τ_i is scheduled for kh_iC_i time units in $\mathcal{S}_k$, τ_i^{kH} is scheduled for its WCET in $\mathcal{S}^{kH}$. Thus, there exists a HRT-feasible schedule of Γ^{kH}. □

B. Sufficient Condition for SRT-Feasibility

In this section, we give a sufficient SRT-feasibility condition for gang tasks as shown in Lemma 4.

Lemma 4. *If there is a periodic task system Γ^{kH} that is HRT-feasible, then the sporadic task system Γ is SRT-feasible.*

Note that we do not require the same value of k in both Lemmas 3 and 4. To prove Lemma 4, we give a server-based scheduling policy for Γ based on a HRT-feasible schedule of Γ^H. For ease of notation, we prove the lemma for Γ^H. We begin by defining reservation servers.

Reservation servers. For each task τ_i, we define a periodic *reservation server* S_i^H. We denote the set of all servers as Γ_s^H. Each server S_i^H has a period $T_i^H = H$, a *horizontal budget* $C_i^H = h_iC_i$, and a degree of parallelism of $m_i^H = m_i$. The *total budget*, also the called *budget*, of S_i^H is $m_iC_i^H$. Thus, by Def. 2 (with $k = 1$), S_i^H and τ_i^H have the same period and degree of parallelism.

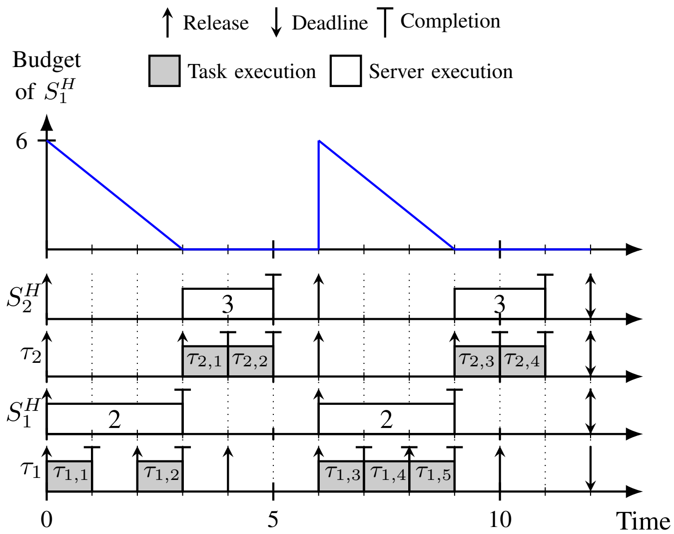


Fig. 2: Example server-based scheduling. The numbers inside server execution boxes denote m_i values.

Replenishment Rule. For any non-negative integer ℓ, the budget of S_i^H is replenished to $m_i C_i^H$ at time $\ell \cdot H$.

Consumption Rule. S_i^H consumes budget at the rate of m_i execution units per unit of time when it is scheduled until its budget is exhausted.

Scheduling servers. Servers are scheduled according to the following rule.

P Let $\mathcal{S}^H$ be a HRT schedule of Γ^H where each job of any task Γ^H executes for its WCET. The servers in Γ_s^H are scheduled according to $\mathcal{S}^H$, *i.e.*, server S_i^H is scheduled at time t if and only if τ_i^H is scheduled at time t in $\mathcal{S}^H$.

Ex. 1. Assume that Γ consists of two gang tasks $\tau_1 = (2, 1, 2)$ and $\tau_2 = (3, 1, 3)$ to be scheduled on a four-core platform. Since the task periods are 2.0 and 3.0, we have $H = 6$, $h_1 = 6/2 = 3$, and $h_2 = 6/3 = 2$. Thus, by Def. 2, Γ^H consists of periodic tasks $\tau_1^H = (0, 6, 3, 2)$ and $\tau_2^H = (0, 6, 2, 3)$. As shown in Fig. 2, there is a HRT-feasible schedule of Γ^H, which, by Rule P, is also the server schedule of S_1^H and S_2^H.

At time 0, the budget of S_1^H is replenished to $m_2 C_2^H = 2 \cdot 3 = 6$. During time $[0, 3)$, S_1^H is scheduled, hence, its budget is consumed at the rate of 2.0 unit per unit of time during $[0, 3)$. Thus, S_1^H's budget is exhausted at time 3. $\Diamond$

For ease of notation, we also denote the server schedule by $\mathcal{S}^H$. By Rule P, we have the following lemma, which shows that S_i^H has sufficient budget to be scheduled whenever τ_i^H is scheduled in $\mathcal{S}^H$.

Lemma 5. *If τ_i^H is scheduled in $\mathcal{S}^H$ at time t, then S_i^H has at least m_i units of budget remaining at time t.*

Proof. Assume that t is the first time instant when S_i^H has less than m_i units of remaining budget, but τ_i^H is scheduled. Let ℓ be the non-negative integer such that $t \in [\ell H, (\ell+1)H)$. By the Replenishment Rule, S_i^H's budget is $m_i C_i^H$ at time ℓH. By the Consumption Rule, S_i^H's budget is consumed at the rate of m_i units per unit of time when it is scheduled. Thus, the remaining budget at time t is an integer multiple of m_i. Therefore, at time t, S_i^H's budget is at most 0.

By the Consumption Rule, S_i^H is scheduled for at least $\frac{m_i C_i^H}{m_i} = C_i^H$ time units during $[\ell H, t)$. Thus, By Rule P, τ_i^H is scheduled for at least C_i^H time units during $[\ell H, t)$. By Def. 2, $\tau_i^H \in \Gamma^H$ releases its $(\ell+1)^{st}$ job at time ℓH. Thus, by the definition of $\mathcal{S}^H$, the $(\ell+1)^{st}$ job of τ_i^H completes by time t, as τ_i^H is scheduled for at least C_i^H time units during $[\ell H, t)$. Since $t < (\ell+1)H$, there is no ready job of τ_i^H at time t. Thus, τ_i^H cannot be scheduled at time t. Contradiction. $\square$

We now prove the following lemma, which we will later use to prove Lemma 4.

Lemma 6. *For any non-negative integer ℓ, server S_i^H is scheduled for $C_i^H = h_i C_i$ time units during any time interval $[\ell H, (\ell+1)H)$ in $\mathcal{S}^H$.*

Proof. By Lemma 5, S_i^H has at least m_i units of remaining budget at any time when τ_i^H is scheduled. Therefore, S_i^H can be scheduled whenever τ_i^H is scheduled.

By Def. 2, $\tau_i^H \in \Gamma^H$ releases its $(\ell+1)^{st}$ job at time ℓH. By the definition of $\mathcal{S}^H$ and Def. 2, the $(\ell+1)^{st}$ job of τ_i^H is scheduled for its WCET of $h_i C_i$ time units and completes by time $(\ell+1)H$ in $\mathcal{S}^H$. Thus, S_i^H is scheduled for $h_i C_i$ time units during $[\ell H, (\ell+1)H)$ in $\mathcal{S}^H$. $\square$

Scheduling tasks on servers. Jobs of the sporadic tasks in Γ are scheduled on servers via the following rules.

R1 Jobs of τ_i are scheduled on server jobs of S_i^H.

R2 If server S_i^H is scheduled and job $\tau_{i,j}$ is ready at time t, then $\tau_{i,j}$ is scheduled on the processors on which S_i^H is scheduled at time t.

Ex. 1 (Cont'd). Consider the scheduling of τ_1 and τ_2 in Fig. 2. At time 0, τ_1 has a ready job $\tau_{1,1}$ and S_1^H is scheduled. Thus, by Rule R2, $\tau_{1,1}$ is scheduled at time 0. At time 1, there is no pending job of task τ_1. Thus, despite S_1^H being scheduled at time 1, no job of τ_1 is scheduled at time 1. $\Diamond$

We now show that each task τ_i has bounded response time if $\mathcal{S}^H$ is a HRT-feasible schedule of Γ^H. We first show, in Lemma 7, that jobs released during $(\ell H, (\ell+1)H]$ complete execution by time $(\ell+2)H$. Using this lemma, we will then derive a response time bound of τ_i in Lemmas 8 and 9.

Lemma 7. *If servers are scheduled according to Rule P and tasks are scheduled according to Rules R1 and R2, then, for any non-negative integer ℓ, any job $\tau_{i,j}$ released during $(\ell H, (\ell+1)H]$ completes execution at or before time $(\ell+2)H$.*

Proof. Assume otherwise. Let ℓ be the smallest non-negative integer such that there is a job $\tau_{i,j}$ released during $(\ell H, (\ell+1)H]$ that completes execution after time $(\ell+2)H$. Thus, any job released during time interval $((\ell-1)H, \ell H]$ completes execution at or before time $(\ell+1)H$. Therefore, no job released at or before time ℓH is pending at or after time $(\ell+1)H$.

Let L be the remaining execution time of τ_i's jobs that are released during $(\ell H, (\ell+1)H]$ at time $(\ell+1)H$. Since τ_i

releases its job sporadically, at most $H/T_i = h_i$ jobs of τ_i are released during $(\ell H, (\ell+1)H]$. Therefore, $L \leq h_i C_i$. By Lemma 6, S_i^H is scheduled for $h_i C_i$ time units during $[(\ell+1)H, (\ell+2)H)$. Since jobs of τ_i execute sequentially and $L \leq h_i C_i$, by Rule R2, all job released during $(\ell H, (\ell+1)H]$ must complete execution by time $(\ell+2)H$, a contradiction. $\square$

Lemma 8. *Let $\tau_{i,j}$ be the q^{th} job of τ_i among τ_i's jobs that are released during $(\ell H, (\ell+1)H]$ where $q \leq h_i$ and ℓ is a non-negative integer. If servers are scheduled according to Rule P and tasks are scheduled according to Rules R1 and R2, then $\tau_{i,j}$'s response time is at most $2H - (h_i - q)C_i - (q-1)T_i$.*

Proof. We first prove that S_i^H is scheduled for at least qC_i time units during $[(\ell+1)H, (\ell+2)H - (h_i - q)C_i)$. Assume otherwise. During $[(\ell+2)H - (h_i-q)C_i, (\ell+2)H)$, S_i^H can be scheduled for at most $(h_i - q)C_i$ time units. Thus, S_i^H is scheduled during $[(\ell+1)H, (\ell+2)H)$ for less than $qC_i + (h_i - q)C_i = h_i C_i$ time units, contradicting Lemma 6.

By Lemma 7, no job released at or before time ℓH is pending after time $(\ell+1)H$. The total execution time of the first q jobs of τ_i released during $(\ell H, (\ell+1)H]$ is at most qC_i. Therefore, by Rule R2, $\tau_{i,j}$ completes execution at or before time $(\ell+2)H - (h_i - q)C_i$, as S_i^H is scheduled for at least qC_i time units during $[(\ell+1)H, (\ell+2)H - (h_i - q)C_i)$.

Since τ_i releases jobs sporadically, we have $r_{i,j} \geq \ell H + (q-1)T_i$. Therefore, $\tau_{i,j}$'s response time is at most $(\ell+2)H - (h_i - q)C_i - \ell H - (q-1)T_i = 2H - (h_i-q)C_i - (q-1)T_i$. $\square$

Lemma 9. *If servers are scheduled according to Rule P and tasks are scheduled according to Rules R1 and R2, then task τ_i's response time is at most $2H - (h_i - 1)C_i$.*

Proof. Let $\tau_{i,j}$ be an arbitrary job of τ_i. Assume that $\tau_{i,j}$ is released during $(\ell H, (\ell+1)H]$ where ℓ is a non-negative integer and $\tau_{i,j}$ is the q^{th} job among τ_i's jobs that are released during $(\ell H, (\ell+1)H]$. By Lemma 8, $\tau_{i,j}$'s response time is at most $2H - (h_i - q)C_i - (q-1)T_i = 2H - (h_i - 1)C_i + (q-1)(C_i - T_i)$. Since $q \geq 1$ and $C_i \leq T_i$, $\tau_{i,j}$'s response time is at most $2H - (h_i - 1)C_i$. Thus, the lemma holds. $\square$

By Lemma 9, Γ is SRT-feasible. This proves Lemma 4.

The response-time bound in Lemma 9 can be exponential with respect to the task count. However, for common task systems that have pseudo-harmonic periods, where $H = T_{max}$ holds, the response-time bound is less than $2T_{max}$.

C. Computational Complexity of SRT-Feasibility

We now prove the following theorem using the conditions derived in Secs. III-A and III-B.

Theorem 1. *SRT-feasibility for gang tasks is NP-hard.*

Proof. The proof is via reduction from the partition problem.

The partition problem. Given a set $A = \{a_1, a_2, \cdots, a_p\}$ of positive integers with $\sum_{i=1}^{p} a_i = 2B$, the partition problem asks whether A can be partitioned into two equal-sum subsets A_1 and A_2, *i.e.*, $\sum_{a \in A_1} = \sum_{a \in A_2} = B$.

Reduction. Let $A = \{a_1, a_2, \cdots, a_p\}$ with $\sum_{i=1}^{p} a_i = 2B$ be an arbitrary instance of the partition problem. We construct an instance of the SRT-feasibility problem as follows. Let Γ be a set of p sporadic gang tasks to be scheduled on B processors. Task $\tau_i \in \Gamma$ has a period of 2.0 time units, a WCET of 1.0 time unit, and $m_i = a_i$.

We now prove that A can be partitioned into two equal-sum subsets if and only if there exists a schedule $\mathcal{S}$ of Γ on B processors where each task has bounded tardiness.

Sufficiency. Assume that A can be partitioned into two equal-sum subsets A_1 and A_2. We will prove that Γ^H is HRT-feasible, which, by Lemma 4, implies that each task in Γ has bounded tardiness under some scheduling algorithm. Since each task's period is 2.0, we have $H = 2.0$. Therefore, by Def. 2, Γ^H consists of p tasks such that $\tau_i^H = (0, 2, 1, m_i)$. Since tasks in Γ^H are implicit-deadline periodic tasks, it suffices to show that there exists a schedule such that the first jobs of all tasks in Γ^H complete by time 2.0. We construct such a HRT-feasible schedule $\mathcal{S}^H$ as follows. $\mathcal{S}^H$ schedules tasks corresponding to subset A_1 (respectively, A_2) during time interval $[0, 1)$ (respectively, $[1, 2)$). Since $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i = B$ and $m_i = a_i$ for all i, exactly B processors execute jobs of Γ^H at any time during $[0, 2)$. Since $A_1 \cup A_2 = A$ and $A_1 \cap A_2 = \emptyset$, each task in Γ^H is scheduled for exactly 1.0 time unit in $\mathcal{S}^H$. Thus, in $\mathcal{S}^H$, the first jobs of all tasks in Γ^H complete by time 2.0.

Necessity. Assume that there is a schedule $\mathcal{S}$ of Γ on B processors where each task in Γ has bounded tardiness. Then, by Lemma 3 and Def. 2, there exists a positive integer k and a HRT-feasible task system Γ^{kH} consisting of tasks $\tau_i = (0, 2k, k, m_i)$. Since tasks in Γ^{kH} are periodic and have implicit deadlines, there is a schedule $\mathcal{S}^{kH}$ of Γ^{kH} on B processors where the first jobs of all tasks in Γ^{kH} complete at or before time $2k$, *i.e.*, each task execute for k time units during time interval $[0, 2k)$.

We first show that there is no idle processor in $\mathcal{S}^{kH}$ at any time instant during $[0, 2k)$. Assume otherwise. Since there is at least one idle processor during a unit-sized time interval, the total execution of tasks in Γ^{kH} is at most $2kB - 1$ units. The total execution requirement of the first jobs of tasks in Γ^{kH} is $\sum_{i=1}^{p} k \cdot m_i = k \sum_{i=1}^{p} m_i = k \sum_{i=1}^{p} a_i = 2kB$. Thus, at least one task's first job does not complete execution by time $2k$ in $\mathcal{S}^{kH}$ and $\mathcal{S}^{kH}$ cannot be a HRT-feasible schedule of Γ^{kH}, a contradiction.

We now show that there exists a partition of A into two equal-sum subsets. Let Q be the set of tasks that are scheduled during $[0, 1)$ in $\mathcal{S}^{kH}$. Since all B processors are busy during $[0, 1)$, we have $\sum_{\tau_i^{kH} \in Q} m_i = B$. Let A_1 be the subset of A that consists of elements a_i corresponding to tasks in Q. Thus, $\sum_{a_i \in A_1} a_i = B$. Since $\sum_{a_i \in A} a_i = 2B$, we have $\sum_{a_i \in A \setminus A_1} a_i = B$. Thus, there exists a partition of two equal-sum subsets of A. $\square$

Algorithm 1 FP-scheduling of servers.

```
Variables:
    O : A priority ordering
    Sched(t) : Set of jobs to be scheduled at time t
procedure FP
    M' ← M
    Order servers according to O
    for each S_i^H ∈ Γ^H do
        if m_i ≤ M' and S_i^H's remaining budget > 0 then
            Sched(t) ← Sched(t) ∪ {S_i^H}
            M' ← M' − m_i
```

IV. Schedulability under Server-Based Scheduling

In Sec. III, we gave a server-based approach for scheduling gang tasks. We showed that if servers can be scheduled to meet their deadlines, then the gang tasks in Γ have bounded tardiness under the server-based scheduling policy. However, we relied on a HRT schedule of the servers (Rule P). Unfortunately, obtaining such a schedule is NP-hard in the strong sense [14]. In this section, we provide some scheduling policies and corresponding exact HRT-schedulability tests for servers, which provide a sufficient means of testing SRT-feasibility for gang tasks by Lemma 4.

Referring to the server-based scheme used to prove this lemma, it is important to note that the HRT-schedulability of the servers in Γ_s^H under a given scheduling policy can be different from HRT-schedulability of the tasks in Γ^H under that policy. This is because a server S_i^H is required to be scheduled for exactly C_i^H time units during $[0, H)$, as this ensures that τ_i receives a sufficient processor allocation during $[0, H)$. In contrast, for Γ^H, τ_i^H can execute for less than its WCET C_i^H, which can cause scheduling anomalies by causing some jobs to miss their deadlines. Thus, for servers, only the case where S_i^H is scheduled for exactly C_i^H time units during $[0, H)$ needs to be considered, which may not be sufficient for HRT-schedulability of Γ^H under the same scheduling.

A. *Fixed-Priority Scheduling of Servers*

Under *fixed-priority* (FP) scheduling, each server has a fixed priority. At any time instant, the highest-priority servers that can execute together (without requiring more than M processors) are scheduled as in Alg. 1. As all servers are replenished synchronously every H time units, FIFO and implicit-deadline GEDF scheduling (each with *consistent* tie-breaking) are equivalent to FP when scheduling servers.

Determining server priorities. We consider the following heuristics for determining server priorities.

- **Parallelism-decreasing order.** S_i^H has higher priority than S_j^H if $m_i \geq m_j$, with ties being broken consistently.
- **Utilization-decreasing order.** S_i^H has higher priority than S_j^H if $u_i \geq u_j$, with ties being broken consistently.

Schedulability test. The HRT-schedulability of the servers under FP scheduling can be determined by simulating the server schedule over the time interval $[0, H)$. The time complexity for this is polynomial with respect to the task and processor counts. This is because no server is replenished within $(0, H)$, so the servers are scheduled non-preemptively. Thus, scheduling decisions are taken only at time 0 and when a server exhausts its budget. Hence, there are $O(n)$ time instants when scheduling decisions are made. Further, each such decision is of polynomial time complexity.

B. *Least-Laxity-First Scheduling of Servers*

Under *least-laxity* (LLF) scheduling, servers with smaller *laxity* have higher priority. A server's laxity corresponds to the amount of time it can be delayed without violating its deadline. Formally, for a server S_i^H, letting $C_i^H(t)$ (resp., $D_i^H(t)$) to denote its remaining budget (resp., remaining time to its deadline) at time t, its laxity $L_i^H(t)$ at time t is $L_i^H(t) = D_i^H(t) - C_i^H(t)$. Thus, LLF scheduling also functions like Alg. 1 with $\mathcal{O}$ denoting LLF ordering.

Schedulability test. Similar to FP scheduling, the HRT-schedulability of servers under LLF scheduling can be done by simulating the server schedule during $[0, H)$. However, unlike FP scheduling, the simulation may take $O(H)$ time, as server priorities may change during runtime.

C. *ILP-Based Scheduling of Servers*

Finally, we show that a server schedule can be obtained by solving an integer linear program (ILP), specified as follows.

Variables. For each server S_i^H, we define H variables $x_{i,1}^H, x_{i,2}^H, \ldots, x_{i,H}^H$. $x_{i,t}^H$ is 1 if S_i^H is scheduled during time interval $[t-1, t)$ and 0 otherwise.

Constraint 1. S_i^H is scheduled for $h_i C_i$ time units (its horizontal budget—see the discussion after Lemma 4) in $[0, H)$:

$$\forall i :: \sum_{t=1}^{H} x_{i,t}^H = h_i C_i.$$

Constraint 2. At most M processors are occupied at any time:

$$\forall t :: \sum_{i=1}^{n} m_i \cdot x_{i,t}^H \leq M.$$

Note that m_i is a constant.

Translating from a valid assignment of values to the $x_{i,t}^H$ variables to a correct server schedule is straightforward. Note that this method provides an exact sever feasibility test. Unfortunately, it has exponential time complexity.

V. Schedulability under GEDF

In this section, we consider the preemptive scheduling of gang tasks by GEDF, which functions as shown in Alg. 2. Under GEDF, ready jobs with earlier deadlines have higher priorities. We assume that deadline ties are broken arbitrarily but consistently (*e.g.*, by task index). When considering a ready job $\tau_{i,j}$ under GEDF, if m_i is larger than the number of remaining available processors, then $\tau_{i,j}$ is skipped (line 5 in Alg. 2).

Algorithm 2 GEDF job selection policy.

Variables:
Ready(t) : Set of ready jobs at time t
Sched(t) : Set of jobs to be scheduled at time t

```
procedure GEDF
    M' ← M
    Order jobs in Ready(t) in deadline-increasing order
    for each τ_{i,j} ∈ Ready(t) do
        if m_i ≤ M' then
            Sched(t) ← Sched(t) ∪ {τ_{i,j}}
            M' ← M' − m_i
```

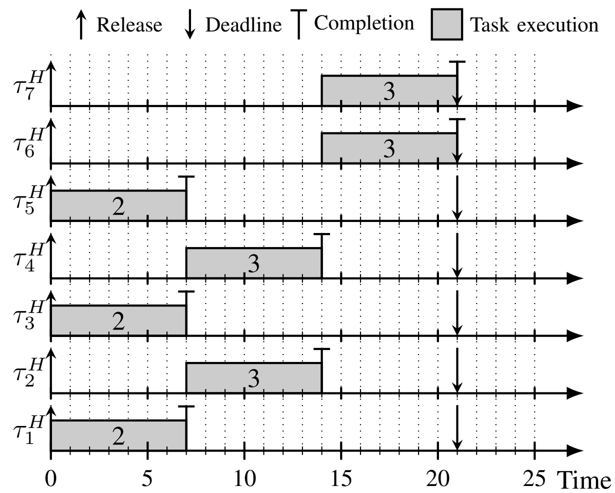


Fig. 3: A HRT-feasible schedule of Γ^H in Theorem 2. The numbers inside execution boxes denote m_i values.

A. Non-SRT-Optimality under GEDF

In this section, we show that GEDF is non-optimal in scheduling SRT gang tasks.

Theorem 2. *GEDF is non-SRT-optimal for gang scheduling.*

Proof. We give a gang task system and a release sequence for it that has unbounded tardiness under GEDF. Let Γ be a gang task system consisting of seven tasks to be scheduled on six processors. Each task τ_i has a WCET of 7.0 time units and a period of 21.0 time units. Let $m_1 = m_3 = m_5 = 2$ and $m_2 = m_4 = m_6 = m_7 = 3$.

Feasibility. Consider Γ^H from by Def. 2. Since all task periods are 21, each τ_i^H has the same period, WCET, and degree of parallelism as τ_i. Fig. 3 shows a HRT-feasible schedule of Γ^H. Thus, by Lemma 4, Γ is SRT-feasible.

Unschedulability under GEDF. Fig. 4 shows a GEDF schedule for Γ where each task τ_i releases its first job at time $i-1$ and subsequent jobs periodically, *i.e.*, its j^{th} job is released at time $i-1+(j-1)T_i$. The GEDF prioritization policy causes at least one idle processor during $[0, 21)$, as a task with $m_i = 3$ is scheduled alongside a task with $m_i = 2$. A similar scenario occurs during the time interval $[22, 43)$. This causes the response time of the second job of each task to be larger than its first job. At times 49 and 50, the third jobs of τ_1 and τ_2, respectively, are scheduled. Thus, the schedule during time $[1, 50)$ starts to repeat at time 50, causing each task's response times to grow unboundedly.

Since GEDF cannot ensure bounded tardiness for a SRT-feasible task system, it is not SRT-optimal for gang tasks. $\square$

Note that, according to the proof of Theorem 2, GEDF is non-SRT-optimal for gang scheduling even when each task's m_i value is at most three.

B. A GEDF Schedulability Test

We now give a schedulability test for GEDF. We begin by introducing some terminology.

Allocation. The cumulative processor capacity allocated to a job $\tau_{i,j}$, task τ_i, task system Γ, and a set of jobs Ψ, in a schedule $\mathcal{S}$ over an interval $[t, t')$ is denoted by $\mathsf{A}(\tau_{i,j}, t, t', \mathcal{S})$, $\mathsf{A}(\tau_i, t, t', \mathcal{S})$, $\mathsf{A}(\Gamma, t, t', \mathcal{S})$, and $\mathsf{A}(\Psi, t, t', \mathcal{S})$, respectively. Thus, $\mathsf{A}(\tau_i, t, t', \mathcal{S}) = \sum_j \mathsf{A}(\tau_{i,j}, t, t', \mathcal{S})$, $\mathsf{A}(\Gamma, t, t', \mathcal{S}) = \sum_{i=1}^{n} \mathsf{A}(\tau_i, t, t', \mathcal{S})$, and $\mathsf{A}(\Psi, t, t', \mathcal{S}) = \sum_{\tau_{i,j} \in \Psi} \mathsf{A}(\tau_{i,j}, t, t', \mathcal{S})$.

Ideal schedule. Let $\widehat{\pi}_1, \widehat{\pi}_2, \ldots, \widehat{\pi}_n$ be n processors with speeds $u_1, u_2, \ldots, u_n$, respectively. In an *ideal schedule* $\mathcal{I}$, each task τ_i is partitioned to execute on processor $\widehat{\pi}_i$. Each job starts execution as soon as it is released and completes execution by its deadline in $\mathcal{I}$. For task τ_i (resp., task system Γ), $\mathsf{A}(\tau_i, t, t', \mathcal{I})) \leq u_i(t'-t)$ (resp., $\mathsf{A}(\Gamma, t, t', \mathcal{I}) \leq U(t'-t)$). In $\mathcal{I}$, parallelism constraints of gang tasks may not be maintained.

lag and LAG. The lag of job $\tau_{i,j}$ at time t in a schedule $\mathcal{S}$ is

$$\mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) = \mathsf{A}(\tau_{i,j}, 0, t, \mathcal{I}) - \mathsf{A}(\tau_{i,j}, 0, t, \mathcal{S}). \quad (1)$$

The lag of a task τ_i at time t in a schedule $\mathcal{S}$ is

$$\mathsf{lag}(\tau_i, t, \mathcal{S}) = \sum_j \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) = \mathsf{A}(\tau_i, 0, t, \mathcal{I}) - \mathsf{A}(\tau_i, 0, t, \mathcal{S}). \quad (2)$$

Since $\mathsf{lag}(\tau_i, 0, \mathcal{S}) = 0$, for $t' \geq t$ we have

$$\mathsf{lag}(\tau_i, t', \mathcal{S}) = \mathsf{lag}(\tau_i, t, \mathcal{S}) + \mathsf{A}(\tau_i, t, t', \mathcal{I}) - \mathsf{A}(\tau_i, t, t', \mathcal{S}). \quad (3)$$

The LAG of a task system Γ in a schedule $\mathcal{S}$ at time t is

$$\mathsf{LAG}(\Gamma, t, \mathcal{S}) = \sum_{\tau_i \in \Gamma} \mathsf{lag}(\tau_i, t, \mathcal{S}) = \mathsf{A}(\Gamma, 0, t, \mathcal{I}) - \mathsf{A}(\Gamma, 0, t, \mathcal{S}). \quad (4)$$

Similarly, the LAG of a set of jobs Ψ is

$$\begin{aligned}\mathsf{LAG}(\Psi, t, \mathcal{S}) &= \sum_{\tau_{i,j} \in \Psi} \mathsf{lag}(\tau_{i,j}, t, \mathcal{S}) \\ &= \sum_{\tau_{i,j} \in \Psi} (\mathsf{A}(\tau_{i,j}, 0, t, \mathcal{I}) - \mathsf{A}(\tau_{i,j}, 0, t, \mathcal{S})). \quad (5)\end{aligned}$$

Since $\mathsf{LAG}(\Psi, 0, \mathcal{S}) = 0$, for $t' \geq t$ we have

$$\mathsf{LAG}(\Psi, t', \mathcal{S}) = \mathsf{LAG}(\Psi, t, \mathcal{S}) + \mathsf{A}(\Psi, t, t', \mathcal{I}) - \mathsf{A}(\Psi, t, t', \mathcal{S}). \quad (6)$$

Ex. 2. Consider three gang tasks $\tau_1 = (8, 5, 3), \tau_2 = (10, 4, 5)$, and $\tau_3 = (12, 7, 2)$ to be scheduled on six processors. Figs. 5 and 6 show an ideal schedule $\mathcal{I}$ and a GEDF schedule $\mathcal{S}$, respectively, of these tasks. Since τ_2's utilization is $(5 \cdot 4)/10 = 20/10 = 2$, it executes at a rate of 2.0 in $\mathcal{I}$. Task τ_2 receives an allocation of $1 \cdot 5 = 5$ (resp., $6 \cdot 2 = 12$) units during $[0, 6)$ in $\mathcal{S}$ (resp., $\mathcal{I}$). Therefore, $\mathsf{lag}(\tau_2, 6, \mathcal{S}) = 12 - 5 = 7$. $\diamond$

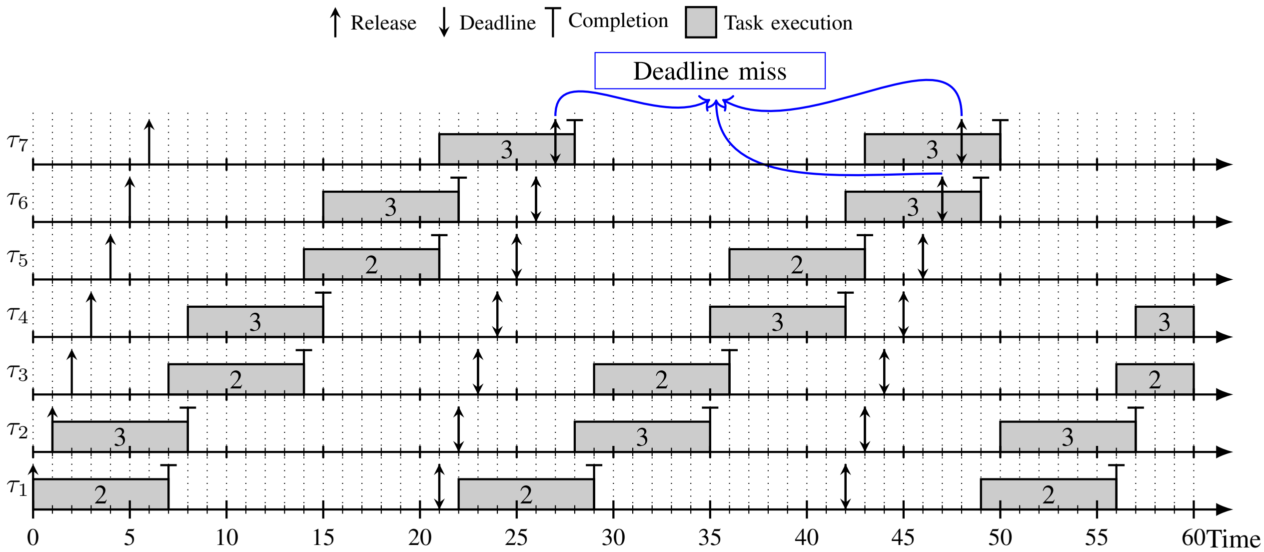


Fig. 4: GEDF schedule of Γ in Theorem 2. The numbers inside execution boxes denote m_i values.

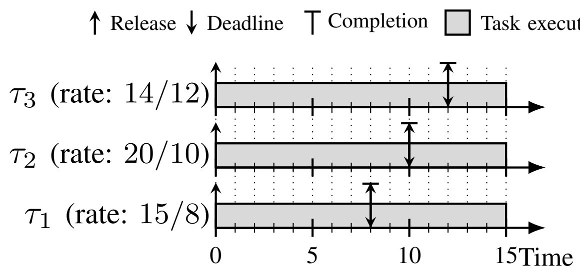


Fig. 5: An ideal schedule.

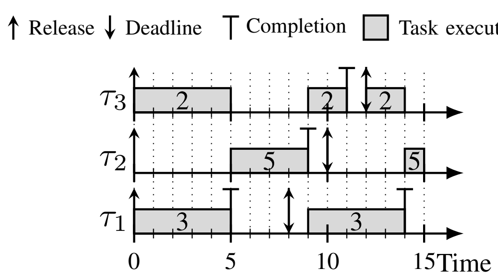


Fig. 6: A GEDF schedule. The numbers inside execution boxes denote m_i values.

Def. 3. *For a task τ_i, let Δ_i denote the maximum possible number of idle processors at any time instant when τ_i has a ready job that cannot execute. Let $\Delta_{max} = \max_i\{\Delta_i\}$.*

Ex. 3. Consider four gang tasks with $m_1 = 3, m_2 = 4, m_3 = 5$, and $m_4 = 6$ to be scheduled on ten processors. If τ_2 has a pending job at time t that cannot execute, then at least seven processors are busy at time t. No combination of other tasks can occupy exactly seven processors. However, if τ_1 and τ_3 execute on $m_1 + m_3 = 8$ processors at time t, then τ_3 cannot execute at time t. Thus, the maximum number of idle processors when τ_3 cannot execute is $\Delta_2 = 10 - 8 = 2$. $\Diamond$

Dong *et al.* gave an $O(M^2 n)$ time dynamic-programming algorithm to compute the Δ_i value of a task [8]. Using Δ_{max}, they established the following sufficient condition for bounded tardiness of gang tasks under GEDF.

Theorem 3 ([8]). *If $U \le M - \Delta_{max}$, then each task in Γ has bounded tardiness under GEDF.*

Consider the task system shown in Fig. 1. Since $M = 4$, $m_1 = 3$, and $m_2 = 2$, we have $\Delta_1 = 2$ and $\Delta_2 = 1$. The utilization of the task system is $\frac{3\cdot 2}{8} + \frac{2\cdot 6}{8} = \frac{18}{8} = 2.25$, which is larger than $M - \Delta_{max} = 4 - 2 = 2$. Thus, the system is not SRT-schedulable under GEDF by Theorem 3.

In this paper, we give an improved sufficient condition for the SRT-schedulability of gang tasks under GEDF. We first introduce some necessary terms.

Def. 4. *Let M_p denote the minimum possible number of busy processors whenever at least p tasks in Γ have pending jobs.*

Ex. 3 (Cont'd). Assume that at least three tasks have pending jobs at time t. If both τ_1 and τ_2 have pending jobs at time t, then at least seven processors are busy at time t under GEDF. On the other hand, if one of τ_1 and τ_2 has no pending job at time t, then both τ_3 and τ_4 have pending jobs at time t. In that case, at least $m_1 + m_3 = 8$ processors are busy at time t under GEDF. Thus, the minimum number of busy processors when at least three tasks have pending job is $M_3 = 7$. $\Diamond$

Before showing how to compute M_p, we first give a sufficient SRT-schedulability condition for GEDF. This condition is comprised of two sub-conditions, which we define next.

Def. 5. *Let $U^b = \sum_{b \text{ smallest}} u_i$, i.e., U^b denotes the sum of the b smallest u_i values.*

$$\exists b \in \mathbb{N}_0 : b < n \wedge U \le (M - \Delta_{max} + U^b) \wedge U \le M_{n-b} \quad (7)$$

$$\forall i : \lambda_i \le 1 \wedge m_i \le M \quad (8)$$

Specifically, we will prove the following theorem.

Theorem 4. *If (7) and (8) hold, then τ_i's tardiness is at most $x + C_i$ where*

$$x \ge \max\{0, \frac{\sum_{(n-b-1) \text{ largest}} m_k C_k - C_{min}}{M - \Delta_{max} + U^{b+1} - U}\}. \quad (9)$$

Here, $\sum_{(n-b-1)\ \mathit{largest}} m_k C_k$ *is the sum of the* $n-b-1$ *largest values of* $m_k C_k$ *among all* k*.*

Note that the denominator in (9) is positive if (7) is met. This is because $U^{b+1} > U^b$, implying $M - \Delta_{max} + U^{b+1} > M - \Delta_{max} + U^b \geq U$. Also, if a task system satisfies the schedulability condition in Theorem 3, then it also satisfies (7). This can be shown by considering $b = 0$, for which $U \leq M - \Delta_{max} \leq M_n$ holds . The last inequality holds because by Def. 3, at least $M - \Delta_{max}$ processors are busy if a task has a pending but unscheduled job.

Ex. 4. Consider seven gang tasks to be scheduled on ten processors GEDF. Let $\tau_1 = (10, 1, 9)$ and $\tau_i = (10, 1, 2)$ for all $i > 1$. Thus, $U = \frac{9 \cdot 1}{10} + 6 \times \frac{2 \cdot 1}{10} = \frac{21}{10} = 2.1$. Also, we have $\Delta_1 = 8$, as τ_1 cannot execute if one of the remaining tasks is scheduled. Thus, $\Delta_{max} = 8$ and $M - \Delta_{max} = 2$. Since $U > M - \Delta_{max} = 2$, the system is deemed SRT-unschedulable by Theorem 3.

Now consider the condition in (7). For $b = 1$, $U^b = 2/10 = 0.2$, so $M - \Delta_{max} + U^b = 2 + 0.2 = 2.2$ holds, which is larger than U. Also, at least nine processors are busy at time t if at least $n - 1$ tasks have pending jobs at time t. Thus, $M_{n-1} = 9 > U$, and (7) is satisfied. Therefore, the system is SRT-schedulable by Theorem 4. ◊

We now prove Theorem 4. Our proof strategy is similar to the LAG-based approach pioneered by Devi and Anderson [5] for ordinary sporadic tasks, and later adapted for gang tasks by Dong *et al.* [8]. The LAG-based analysis in [8] relies on determining an upper bound on LAG by considering lag values at the latest time instant t_0 at or before the deadline t_d of a job of interest such that at least $M - \Delta_{max}$ processors are busy during $[t_0, t_d)$. However, this may not capture the "opportunistic" execution of lower-priority jobs (*e.g.*, the execution of τ_3 during $[0, 5)$ in Fig. 6 despite having lower priority than τ_2). By defining t_0 using the number of tasks with pending jobs, instead of busy processors, we can account for such lower-priority job execution. This may result in a larger LAG upper bound at t_0, as more tasks need to be considered, causing larger tardiness bounds.

Assume that (7) and (8) hold for Γ and let b be the smallest non-negative integer for which (7) is met. Let $\mathcal{S}$ be a GEDF schedule of Γ. We consider an arbitrary job $\tau_{i,j}$ and inductively prove that its tardiness is no more than $x + C_i$ in $\mathcal{S}$. Thus, we assume the following.

F The tardiness of each job with higher priority than $\tau_{i,j}$ is at most $x + C_i$ in $\mathcal{S}$.

Let $t_d = d_{i,j}$ and $t_f = f_{i,j}$. We assume that $t_f > t_d$ holds, otherwise $\tau_{i,j}$'s tardiness is 0.

Def. 6. *Let* ψ *be the set of jobs that have higher priority than* $\tau_{i,j}$*. Let* $\Psi = \psi \cup \{\tau_{i,j}\}$*.*

Under GEDF scheduling, $\tau_{i,j}$ can only be delayed by the jobs in $\Psi \setminus \{\tau_{i,j}\}$. Note that jobs not in Ψ can be scheduled before $\tau_{i,j}$ due to the lack of enough processors to schedule $\tau_{i,j}$ (line 5 in Alg. 2). However, such jobs will be preempted (if needed) as soon as there are enough processors to schedule $\tau_{i,j}$. The following lemma gives an upper bound on lag values.

Lemma 10 ([8]). *For any task* τ_k *and a time instant* $t \leq t_d$*,* $\mathsf{lag}(\tau_k, t, \mathcal{S}) \leq m_k(x\lambda_k + C_k)$ *holds.*

Def. 7. *A time instant* t *is called* b-busy *if at least* $n - b$ *tasks have pending jobs (hence, at most* b *tasks have no pending jobs) in* Ψ *at* t*, and* b-non-busy *otherwise. A time interval is called* b*-busy (resp.,* b*-non-busy) if each instant in the interval is* b*-busy (resp.,* b*-non-busy).*

The number of busy processors in a b-busy time instant t depends on the m_i values and the deadlines of the pending jobs at time t. Also, the number of busy processors may vary throughout a busy interval because of job releases and completions. However, by Def. 4, the number of busy processors is lower bounded by M_{n-b} at any busy instant, as there are at least $n - b$ pending jobs.

Def. 8. *Let* t_0 *be the first time instant such that* $[t_0, t_d)$ *is a* b*-busy interval. Let* τ^* *be the set of tasks with jobs in* Ψ *that are pending at time* $t_0 - 1$*. Note that* $\tau^* = \emptyset$*, if* $t_0 = 0$*.*

Using Lemma 10, we upper-bound the LAG of Ψ at t_0.

Lemma 11. $\mathsf{LAG}(\Psi, t_0, \mathcal{S}) \leq \sum_{\tau_k \in \tau^*} (m_k (x\lambda_k + C_k))$.

Proof. By (5), we have

$$\begin{aligned}\mathsf{LAG}(\Psi, t_0, \mathcal{S}) &= \sum_{\tau_{k,\ell} \in \Psi} \mathsf{lag}(\tau_{k,\ell}, t_0, \mathcal{S}) \\ &= \sum_{\tau_k \in \Gamma} \sum_{\tau_{k,\ell} \in \Psi} \mathsf{lag}(\tau_{k,\ell}, t_0, \mathcal{S}) \\ &= \sum_{\tau_k \in \tau^*} \sum_{\tau_{k,\ell} \in \Psi} \mathsf{lag}(\tau_{k,\ell}, t_0, \mathcal{S}) \\ &\quad + \sum_{\tau_k \notin \tau^*} \sum_{\tau_{k,\ell} \in \Psi} \mathsf{lag}(\tau_{k,\ell}, t_0, \mathcal{S}). \quad (10)\end{aligned}$$

We now prove two claims motivated by (10).

Claim 11.1. *For any* $\tau_k \notin \tau^*$*,* $\sum_{\tau_{k,\ell} \in \Psi} \mathsf{lag}(\tau_{k,\ell}, t_0, \mathcal{S}) \leq 0$*.*

Proof. Since $\tau_k \notin \tau^*$, τ_k has no pending job in Ψ at time $t_0 - 1$. Let $\tau_{k,p} \in \Psi$ be the latest job of τ_k released at or before time $t_0 - 1$ (hence, before time t_0). By the definition of $\mathcal{I}$, for each job $\tau_{k,\ell} \in \Psi$ with $\ell \leq p$, we have $\mathsf{A}(\tau_{k,\ell}, 0, t_0, \mathcal{I}) \leq C_{k,\ell}$. Additionally, since $\tau_{k,p}$ completes execution by time t_0 in $\mathcal{S}$, we have $\mathsf{A}(\tau_{k,\ell}, 0, t_0, \mathcal{S}) = C_{k,\ell}$ for each such job $\tau_{k,\ell}$. Thus, for all $\ell \leq p$, we have $\mathsf{A}(\tau_{k,\ell}, 0, t_0, \mathcal{I}) - \mathsf{A}(\tau_{k,\ell}, 0, t_0, \mathcal{S}) \leq 0$. Therefore, by (1) we have

$$\forall \ell \leq p : \mathsf{lag}(\tau_{k,\ell}, t_0, \mathcal{S}) \leq 0. \quad (11)$$

No job $\tau_{k,\ell} \in \Psi$ with $\ell > p$ can execute before time t_0 in both $\mathcal{I}$ and $\mathcal{S}$. Thus, for $\ell > p$, we have $\mathsf{A}(\tau_{k,\ell}, 0, t_0, \mathcal{I}) = \mathsf{A}(\tau_{k,\ell}, 0, t_0, \mathcal{S}) = 0$. Thus, we have $\forall \ell > p : \mathsf{lag}(\tau_{k,\ell}, t_0, \mathcal{S}) = 0$. Together with (11), this implies the claim. □

Claim 11.2. *For any* $\tau_k \in \tau^*$ *and job* $\tau_{k,\ell} \notin \Psi$*,* $\mathsf{lag}(\tau_{k,\ell}, t_0, \mathcal{S}) \geq 0$*.*

Proof. Since each task executes sequentially, job $\tau_{k,\ell} \notin \Psi$ cannot execute before all jobs of τ_k in Ψ complete execution. Since τ_k has a pending job in Ψ at time $t_0 - 1$, $\tau_{k,\ell}$ cannot be scheduled at or before time $t_0 - 1$ in $\mathcal{S}$. Thus,

$\mathsf{A}(\tau_{k,\ell}, 0, t_0, \mathcal{S}) = 0$ holds. Since $\mathsf{A}(\tau_{k,\ell}, 0, t_0, \mathcal{I}) \geq 0$ holds, by (1), the claim follows. $\square$

By Claim 11.1 and (10), we have $\mathsf{LAG}(\Psi, t_0, \mathcal{S}) \leq \sum_{\tau_k \in \tau^*} \sum_{\tau_{k,\ell} \in \Psi} \mathsf{lag}(\tau_{k,\ell}, t_0, \mathcal{S})$, which is at most $\sum_{\tau_k \in \tau^*} (\sum_{\tau_{k,\ell} \in \Psi} \mathsf{lag}(\tau_{k,\ell}, t_0, \mathcal{S}) + \sum_{\tau_{k,\ell} \notin \Psi} \mathsf{lag}(\tau_{k,\ell}, t_0, \mathcal{S}))$ $= \sum_{\tau_k \in \tau^*} \mathsf{lag}(\tau_k, t_0, \mathcal{S})$, by Claim 11.2. Therefore, by Lemma 10, $\mathsf{LAG}(\Psi, t_0, \mathcal{S}) \leq \sum_{\tau_k \in \tau^*} m_k(x\lambda_k + C_k)$. $\square$

Finally, we give an upper bound on the LAG of Ψ at time t_d.

Lemma 12. $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) \leq \sum_{\tau_k \in \tau^*} (m_k (x\lambda_k + C_k))$.

Proof. Since $[t_0, t_d)$ is a b-busy interval, by Defs. 4 and 7, at least M_{n-b} processors are busy executing jobs in Ψ during $[t_0, t_d)$ in $\mathcal{S}$. Thus, $\mathsf{A}(\Psi, t_0, t_d, \mathcal{S}) \geq M_{n-b}(t_d - t_0)$ holds. By (6), we have

$$\begin{aligned}\mathsf{LAG}(\Psi, t_d, \mathcal{S}) &= \mathsf{LAG}(\Psi, t_0, \mathcal{S}) + \mathsf{A}(\Psi, t_0, t_d, \mathcal{I}) \\ &\quad - \mathsf{A}(\Psi, t_0, t_d, \mathcal{S}) \\ &\leq \{\text{Since } \mathsf{A}(\Psi, t_0, t_d, \mathcal{I}) \leq U(t_d - t_0) \text{ and} \\ &\qquad \mathsf{A}(\Psi, t_0, t_d, \mathcal{S}) \geq M_{n-b}(t_d - t_0)\} \\ &\quad \mathsf{LAG}(\Psi, t_0, \mathcal{S}) + U(t_d - t_0) - M_{n-b}(t_d - t_0) \\ &\leq \{\text{Since } U \leq M_{n-b} \text{ by (7)}\} \\ &\quad \mathsf{LAG}(\Psi, t_0, \mathcal{S}) \\ &\leq \{\text{By Lemma 11}\} \\ &\quad \sum_{\tau_k \in \tau^*} (m_k (x\lambda_k + C_k)).\end{aligned}$$

$\square$

Let W be the total remaining workload of Ψ at time t_d in $\mathcal{S}$. Using Lemma 12, we upper bound W in the lemma below.

Lemma 13. $W \leq \sum_{\tau_k \in \tau^*} (m_k (x\lambda_k + C_i))$.

Proof. By the definition of $\mathcal{I}$, all jobs in Ψ finish execution by time t_d in $\mathcal{I}$. The completed workload of jobs in Ψ at time t_d is $\mathsf{A}(\Psi, 0, t_d, \mathcal{S})$. Thus, the remaining workload of Ψ at time t_d in $\mathcal{S}$ is $\mathsf{LAG}(\Psi, t_d, \mathcal{S}) \leq \sum_{\tau_k \in \tau^*} (m_k (x\lambda_k + C_k))$. $\square$

The following lemma gives a lower bound on W if $\tau_{i,j}$'s tardiness exceeds $x + C_i$. The lemma can be proven by considering an interval $[t_d, t_d + t_y)$ during which at least $M - \Delta_{max}$ processors execute jobs in Ψ.

Lemma 14 ([8])**.** *If $W \leq (M - \Delta_{max})x + C_i$ holds, then $\tau_{i,j}$'s tardiness is at most $x + C_i$.*

The next lemma shows that Theorem 4 holds.

Lemma 15. *$\tau_{i,j}$'s tardiness is at most $x + C_i$.*

Proof. Assume that $\tau_{i,j}$'s tardiness is more than $x + C_i$. Then, by Lemma 14, $W > (M - \Delta_{max})x + C_i$ holds. By Lemma 13, we have

$$(M - \Delta_{max})x + C_i < \sum_{\tau_k \in \tau^*} m_k (x\lambda_k + C_k),$$

which implies

$$x < \frac{\sum_{\tau_k \in \tau^*} m_k C_k - C_i}{M - \Delta_{max} - \sum_{\tau_k \in \tau^*} m_k \lambda_k}$$

Algorithm 3 Finding M_p.

Variables:
- $F[i, \ell, m]$ is initially NULL
- $B[i, \ell, m]$ precomputed true/false values
- m_i values in non-decreasing order

procedure FIND_$M_p(i, \ell, m)$
  if $F[i, \ell, m] \neq$ NULL **then**
    return $F[i, \ell, m]$
  if $\ell < 0 \vee m < 0$ **then**
    return ∞
  if $i = n$ **then**
    $F[i, \ell, m] \leftarrow \infty$
    if $\ell = 0$ and $m_i \leq m$ **then**
      $F[i, \ell, m] \leftarrow m_i$
    if ($\ell = 0$ and $m_i > m$) or $\ell = 1$ **then**
      $F[i, \ell, m] \leftarrow 0$
    return $F[i, \ell, m]$
  $x_1 \leftarrow$ FIND_$M_p(i + 1, \ell - 1, m)$
  $x_2 \leftarrow$ FIND_$M_p(i + 1, \ell, m - m_i) + m_i$
  for each $x \in \{m - m_i + 1, \cdots, m\}$ **do**
    if $B[i + 1, \ell, x] = true$ **then**
      $x_3 \leftarrow x$
      break
  $F[i, \ell, m] \leftarrow \min(x_1, x_2, x_3)$
  return $F[i, \ell, m]$

$$\begin{aligned}&= \frac{\sum_{\tau_k \in \tau^*} m_k C_k - C_i}{M - \Delta_{max} - \sum_{\tau_k \in \tau^*} u_k} \\ &\leq \{\text{Since } |\tau^*| \leq n - b - 1 \text{ and } C_i \geq C_{min}\} \\ &\quad \frac{\sum_{(n-b-1) \text{ largest}} m_k C_k - C_{min}}{M - \Delta_{max} - \sum_{(n-b-1) \text{ largest}} u_k} \\ &= \frac{\sum_{(n-b-1) \text{ largest}} m_k C_k - C_{min}}{M - \Delta_{max} - U + \sum_{(b+1) \text{ smallest}} u_k} \\ &= \{\text{By Def. 5}\} \\ &\quad \frac{\sum_{(n-b-1) \text{ largest}} m_k C_k - C_{min}}{M - \Delta_{max} + U^{b+1} - U},\end{aligned}$$

which contradicts (9). $\square$

Discussion. The tardiness bound given in Theorem 4 is smaller for large b values. Thus, to compute a small tardiness bound, the largest b value that satisfies (7) should be picked.

Computing M_p. We now show how to compute the value of M_p. We begin by giving the following property.

Property 1. *Let M_p^e denote the the minimum possible number of busy processors whenever* exactly *p tasks in Γ have pending jobs. Then, for any $p < n$, $M_p^e \leq M_{p+1}^e$.*

By Property 1, we have $M_p = M_p^e$. Thus, we compute M_p by determining M_p^e. To compute M_p^e, we first index tasks in the non-decreasing order by m_i, *i.e.*, $m_i \leq m_{i+1}$.

Property 2. *If M' processors are busy at time t, then, for any unscheduled task τ_i with pending jobs, $M' + m_i > M$ holds.*

We give a dynamic-programming algorithm to compute M_p^e that satisfies Property 2 as shown Alg. 3. Alg. 3 uses a precomputed array B, which we compute via dynamic programming. We first describe how B is computed.

Let $B[i,\ell,m]$ be *true* if there exists a subset $\Gamma_{i,\ell}$ of $(n-i+1)-\ell$ tasks with pending jobs (*i.e.*, exactly ℓ tasks with no pending jobs) in $\{\tau_i,\tau_{i+1},\cdots,\tau_n\}$ such that $(\exists\, \Gamma^e_{i,\ell}\subseteq\Gamma_{i,\ell}:\sum_{\tau_k\in\Gamma^e_{i,\ell}} m_k=m)$ holds, and *false* otherwise. Informally, if $B[i,\ell,m]$ is true and if there are exactly ℓ tasks in $\{\tau_i,\tau_{i+1},\cdots,\tau_n\}$ that have no pending job at time t, then there is a way to select jobs from the remaining $(n-i+1)-\ell$ tasks to occupy exactly m processors.

We can compute the array B in $O(n^2M)$ time via dynamic programming using the following recurrence.

$$B[i,\ell,m]=\begin{cases} true & \text{if } i=n\,\wedge \\ & \quad((m=m_n\wedge\ell=0) \\ & \quad\vee(m=0\wedge\ell\le 1)) \\ false & \text{if } \ell<0\vee(i=n\,\wedge \\ & \quad((m=m_n\wedge\ell\neq 0) \\ & \quad\vee(m=0\wedge\ell>1) \\ & \quad\vee(m\notin\{0,m_n\}))) \\ B[i+1,\ell,m-m_i] & \\ \quad\vee\, B[i+1,\ell,m] & \text{otherwise} \\ \quad\vee\, B[i+1,\ell-1,m] & \end{cases} \tag{12}$$

The first two cases in (12) cover the base cases. For $i=n$, $B[n,\ell,m]$ is only true if $\ell=0$ (τ_n has a pending job) and $m=m_n$, or $\ell\le 1$ and $m=0$. For $i<n$, $B[i,\ell,m]$ is computed via the third case in (12). $B[i+1,\ell,m-m_i]$ (resp., $B[i+1,\ell,m]$) holds when τ_i has a pending job that executes on m_i processors (resp., does not execute), ℓ tasks in $\{\tau_{i+1},\cdots,\tau_n\}$ have no pending jobs, and those tasks with pending jobs occupy $m-m_i$ (resp., m) processors. $B[i+1,\ell-1,m]$ holds when τ_i has no pending job, $\ell-1$ tasks in $\{\tau_{i+1},\cdots,\tau_n\}$ have no pending jobs, and the tasks with pending jobs occupy m processors.

Using the array B, procedure $\text{FIND\_}M_p(i,\ell,m)$ in Alg. 3 determines the minimum number of busy processors when $\{\tau_i,\tau_{i+1},\cdots,\tau_n\}$ are scheduled on m processors and ℓ tasks among them have no pending jobs. To compute M_p, we invoke $\text{FIND\_}M_p(1,n-p,M)$. We now describe Alg. 3. Lines 2–3 check whether the subproblem is already computed and lines 4–12 cover the base cases. Line 13 makes a recursive call to determine the minimum number of busy processors when τ_i has no pending job (thus, $\ell-1$ tasks among $\{\tau_{i+1},\ldots,\tau_n\}$ have no pending jobs). Line 14 considers the case when τ_i has a pending and scheduled job (thus, ℓ tasks among $\{\tau_{i+1},\ldots,\tau_n\}$ have no pending jobs), Lines 15–18 consider the case when τ_i has pending but unscheduled jobs. In this case, at least $m-m_i+1$ processors must be busy. Thus, for each $x\in\{m-m_i+1,\cdots,m\}$, we consult the array B to determine the lowest possible x value for which $B[i+1,\ell,x]$ is true. Note that if $B[i+1,\ell,x]$ is true, then any unscheduled task τ_k with $k>i$ satisfies Property 2 because $m_i\le m_k$. Finally, the minimum among the three cases are returned.

Since $i\le n$, $\ell\le n$, and $m\le M$ holds, $\text{FIND\_}M_p$ is called at most $O(n^2M)$ times. In each call, lines 15–18 take $O(M)$ time with the precomputed B array. Thus, the total time to compute M_p is $O(n^2M^2)$. Since M_p can be computed in polynomial time, (7) can also be checked in polynomial time.

VI. Experiments

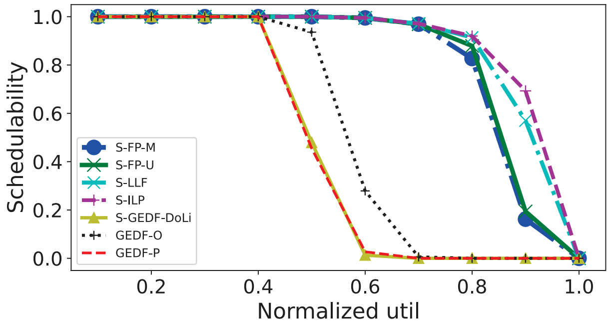
In this section, we provide the results of a schedulability study we conducted to evaluate our proposed approaches.

Our task-system generation method is similar to that used in prior gang-task-related schedulability studies [6]–[8]. We generated task systems randomly for systems with $M=16$ or $M=32$ processors. Motivated by automotive use cases, we chose task periods from $\{2,5,10,20,50,100,200,1000\}$ms [13]. We considered *light*, *medium*, or *heavy* horizontal task utilizations, which are uniformly distributed in $[0.01,0.1]$, $[0.1,0.3]$, and $[0.3,1]$, respectively. We set each task's WCET C_i to $T_i\cdot u_i$ rounded to the next microsecond. We considered *small*, *moderate*, or *heavy* degrees of parallelism, for which m_i values were uniformly distributed in $[1,\frac{M}{4}]$, $[\frac{M}{4},\frac{5M}{8}]$, and $[\frac{5M}{8},\frac{7M}{8}]$, respectively. We varied the *normalized utilization*, *i.e.*, U/M, from 0.1 to 1.0 with a step size of 0.1. For each combination of M, horizontal task utilization, degree of parallelism, and normalized utilization, we generated 1,000 task systems. We generated each such task system by creating tasks until the system's normalized utilization exceeds the desired value, and by then reducing the last task's utilization so that the normalized utilization equals the desired value. We call each combination of M, horizontal task utilization, and degree of parallelism a *scenario*.

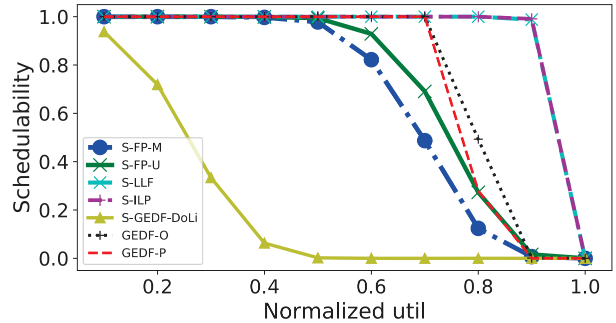
We assessed the SRT-schedulability of each task system under both GEDF and the server-based scheduling policies given in Sec. IV. For scheduling servers, we considered FP scheduling with parallelism-decreasing priorities (S-FP-M), FP scheduling with utilization-decreasing priorities (S-FP-U), LLF scheduling (S-LLF), and ILP-based scheduling (S-ILP). To assess the efficacy of the schedulability tests of servers given in Sec. IV, we also determined the schedulability of servers under GEDF by methods in [6] (S-ILP). For GEDF scheduling of gang tasks, we determined schedulability by the **p**rior method (denoted GEDF-P) from Dong *et al.*, *i.e.*, Theorem 3, and by **o**ur method (denoted GEDF-O), *i.e.*, Theorem 4. For each scenario, we computed *acceptance ratios*, which give the percentage of task systems that were schedulable under each approach. We present a representative selection of our results in Fig. 7.

Observation 1. *In all scenarios,* S-LLF *had a higher acceptance ratio than* S-FP-M, S-FP-U, GEDF-O, *and* GEDF-P. *For most scenarios,* S-FP-M *and* S-FP-U *had higher acceptance ratios than* GEDF-P. *The average improvement in* S-LLF, S-FP-M, S-FP-U, *and* GEDF-O *over* GEDF-P *was 37.65%, 26.37%, 28.79%, and 8.32%, respectively.*

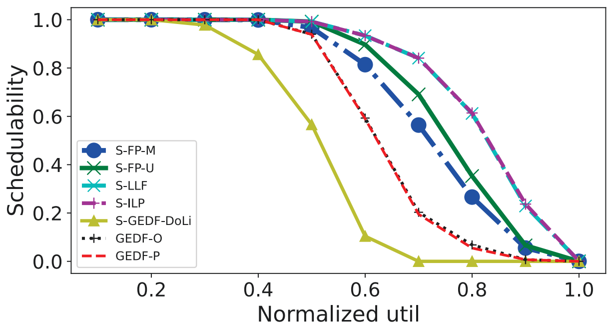
This can be seen in Fig. 7(a)–(c). Being a dynamic-priority scheduling algorithm, LLF can schedule more task systems than the other approaches. In most scenarios, server-based FP scheduling outperformed GEDF, while utilization-decreasing priority ordering outperformed parallelism-decreasing priority



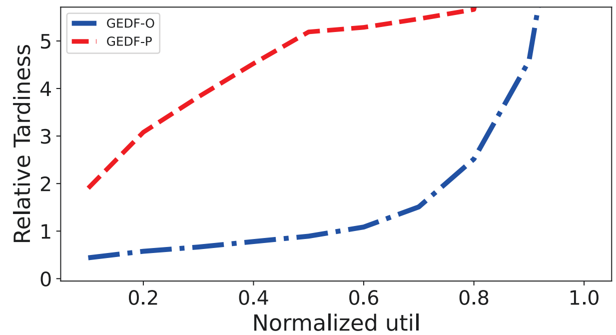
(a) Acceptance ratio for $M = 32$, medium horizontal utilizations, and moderate degree of parallelism.



(b) Acceptance ratio for $M = 32$, heavy horizontal utilizations, and small degree of parallelism.



(c) Acceptance ratio for $M = 16$, heavy horizontal utilizations, and moderate degree of parallelism.



(d) Relative tardiness bounds under GEDF for $M = 16$, heavy horizontal utilizations, and moderate degree of parallelism.

Fig. 7: Experimental results.

ordering. As expected, more task systems were schedulable by Theorem 4 than by Theorem 3.

Observation 2. *For scenarios with a small degree of parallelism,* GEDF-O *and* GEDF-P *had higher acceptance ratios than* S-FP-M *and* S-FP-U.

This can be seen in Fig. 7(b). When the m_i values are small, Δ_{max} (Def. 3) is also small. This causes more task systems to be schedulable under GEDF by Theorem 3.

Observation 3. *The average improvement in* S-ILP *over* S-LLF, S-FP-M, S-FP-U, GEDF-O, *and* GEDF-P *was 0.16%, 9.10%, 7.05%, 27.29%, and 37.88%, respectively. In all scenarios,* S-GEDF-DoLi *had smaller acceptance ratios than* S-ILP, S-LLF, S-FP-M, *and* S-FP-U.

Figs. 7(a)–(c) show this. Server-based LLF scheduling scheduled most task systems that were schedulable under ILP-based scheduling. In contrast, many SRT-feasible task systems were deemed unschedulable by the other considered approaches. S-GEDF-DoLi was deemed more pessimistic than S-ILP, S-LLF, S-FP-M, and S-FP-U, as it is applicable to HRT-scheduling of sporadic gang tasks.

To compare the tardiness bounds derived under GEDF (Theorem 4) with those from [8], we computed *relative tardiness bounds* for all task systems that are SRT-schedulable according to the corresponding GEDF schedulability tests. A task's relative tardiness is computed by dividing its tardiness by the maximum period, *i.e.*, T_{max}. When computing relative tardiness bounds using Theorem 4, we selected the largest value of b for which (7) was satisfied.

Observation 4. *On average, relative tardiness bounds in* GEDF-O *were 47.53% smaller than* GEDF-P.

Fig. 7(d) shows this. This improvement was due to frequently observed large b values that contributed to a less pessimistic accounting of carry-on workloads.

VII. Conclusion

In this paper, we have considered the SRT-feasibility problem for systems of gang tasks. We have presented a necessary and a sufficient condition for the SRT-feasibility of such systems. Based on these conditions, we have shown that the SRT-feasibility problem for gang task systems is NP-hard. We have also provided server-based scheduling policies for gang tasks and corresponding SRT-schedulability tests for gang tasks based on exact HRT-schedulability tests for the servers. Finally, we have shown that GEDF is non-SRT-optimal for gang tasks and provided a SRT-schedulability test for gang tasks under GEDF. We have provided experimental evaluations that demonstrate the benefits of our approaches.

In future work, we plan to investigate whether the SRT-feasibility problem for gang tasks is NP-hard in the strong sense. We also plan to devise smaller tardiness bounds under GEDF and server-based scheduling. Furthermore, we aim to investigate scheduling policies that may utilize both deadlines and the degree of parallelism in determining job priorities. Finally, we want to study the non-preemptive scheduling of SRT gang tasks, as this is relevant to GPUs as a use case.

References

[1] S. Ahmed and J. Anderson, "Tight tardiness bounds for pseudo-harmonic tasks under global-EDF-like schedulers," in *ECRTS'21*, 2021, pp. 11:1–11:24.

[2] ——, "Exact response-time bounds of periodic DAG tasks under server-based global scheduling," in *RTSS'22*, 2022, pp. 447–459.

[3] W. Ali, R. Pellizzoni, and H. Yun, "Virtual gang scheduling of parallel real-time tasks," in *DATE'21*, 2021, pp. 270–275.

[4] A. Bhuiyan, K. Yang, S. Arefin, A. Saifullah, N. Guan, and Z. Guo, "Mixed-criticality multicore scheduling of real-time gang task systems," in *RTSS'19*, 2019, pp. 469–480.

[5] U. Devi and J. Anderson, "Tardiness bounds under global EDF scheduling on a multiprocessor," in *RTSS'05*, 2005, pp. 330–341.

[6] Z. Dong and C. Liu, "Analysis techniques for supporting hard real-time sporadic gang task systems," *Real Time Syst.*, vol. 55, no. 3, pp. 641–666, 2019.

[7] ——, "A utilization-based test for non-preemptive gang tasks on multiprocessors," in *RTSS'22*, 2022, pp. 105–117.

[8] Z. Dong, K. Yang, N. Fisher, and C. Liu, "Tardiness bounds for sporadic gang tasks under preemptive global EDF scheduling," *IEEE Trans. Parallel Distributed Syst.*, vol. 32, no. 12, pp. 2867–2879, 2021.

[9] J. Erickson, J. Anderson, and B. Ward, "Fair lateness scheduling: reducing maximum lateness in G-EDF-like scheduling," *Real-Time Systems*, vol. 50, no. 1, pp. 5–47, 2014.

[10] J. Goossens and V. Berten, "Gang FTP scheduling of periodic and parallel rigid real-time tasks," *CoRR*, vol. abs/1006.2617, 2010.

[11] J. Goossens and P. Richard, "Optimal scheduling of periodic gang tasks," *Leibniz Trans. Embed. Syst.*, vol. 3, no. 1, pp. 04:1–04:18, 2016.

[12] S. Kato and Y. Ishikawa, "Gang EDF scheduling of parallel task systems," in *RTSS'09*, 2009, pp. 459–468.

[13] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *WATERS'15*, 2015.

[14] M. Kubale, "The complexity of scheduling independent two-processor tasks on dedicated processors," *Inf. Process. Lett.*, vol. 24, no. 3, pp. 141–147, 1987.

[15] S. Lee, N. Guan, and J. Lee, "Design and timing guarantee for non-preemptive gang scheduling," in *RTSS'22*, 2022, pp. 132–144.

[16] S. Lee, S. Lee, and J. Lee, "Response time analysis for real-time global gang scheduling," in *RTSS'22*, 2022, pp. 92–104.

[17] H. Leontyev and J. Anderson, "Tardiness bounds for FIFO scheduling on multiprocessors," in *ECRTS'07*, 2007, p. 71.

[18] ——, "Generalized tardiness bounds for global multiprocessor scheduling," *Real-Time Systems*, vol. 44, no. 1-3, pp. 26–71, 2010.

[19] C. Liu and J. H. Anderson, "Supporting soft real-time DAG-based systems on multiprocessors with no utilization loss," in *RTSS'10*, 2010, pp. 3–13.

[20] G. Nelissen, J. M. i Igual, and M. Nasri, "Response-time analysis for non-preemptive periodic moldable gang tasks," in *ECRTS'22*, 2022, pp. 12:1–12:22.

[21] P. Richard, J. Goossens, and S. Kato, "Comments on "gang EDF schedulability analysis"," *CoRR*, vol. abs/1705.05798, 2017.

[22] S. Tang and J. Anderson, "Towards practical multiprocessor EDF with affinities," in *RTSS'20*, 2020, pp. 89–101.

[23] S. Tang, S. Voronov, and J. Anderson, "GEDF tardiness: Open problems involving uniform multiprocessors and affinity masks resolved," in *ECRTS'19*, 2019, pp. 13:1–13:21.

[24] N. Ueter, M. Günzel, G. von der Brüggen, and J. Chen, "Hard real-time stationary gang-scheduling," in *ECRTS'21*, vol. 196, 2021, pp. 10:1–10:19.

[25] K. Yang and J. Anderson, "On the soft real-time optimality of global EDF on uniform multiprocessors," in *RTSS'17*, 2017, pp. 319–330.