On the Soft Real-Time Optimality of Global EDF on Multiprocessors: From Identical to Uniform Heterogeneous *

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

Abstract

Under the definition of soft real-time (SRT) correctness that requires deadline tardiness to be bounded, both the preemptive and non-preemptive global EDF (GEDF) schedulers are known to be SRT-optimal on identical multiprocessors. This paper considers the potential extension of these results to uniform heterogeneous multiprocessors. In the preemptive case, it is shown that such an extension is possible for two-processor platforms but unlikely for platforms of more than two processors, unless fundamentally new proof techniques are developed. In the non-preemptive case, it is shown that no work-conserving scheduler, including GEDF, can be SRT-optimal on uniform multiprocessors, even if the number of processors is limited to two.

1 Introduction

The advent of multicore technologies has led to signification recent attention being directed at multiprocessor platforms in the real-time system research community. Perhaps inspired by the optimality of earliest-deadline-first (EDF) scheduling on uniprocessors, global EDF (GEDF) scheduling on multiprocessors has continued to receive much interest. Unfortunately, in hard real-time (HRT) systems, where every deadline must be met, GEDF scheduling is not optimal due to the Dhall Effect [3].

Nonetheless, GEDF scheduling has desirable theoretical properties under a more general interpretation of the term *real-time system* that was suggested by Buttazzo [1]. Under this interpretation, a real-time system is one that is able

> "to provide *bounded response times* to tasks with bounded execution, *in all possible scenarios*. "

Clearly, meeting all deadlines guarantees bounded response times. Alternatively, under the definition of soft real-time (SRT) correctness that requires deadline tardiness[1] to be provably bounded, response times are bounded as well. In work on GEDF scheduling, this definition of SRT was first considered by Devi and Anderson [2].

In this paper, we consider the GEDF scheduling of sporadic task systems under this definition of SRT correctness. We define such a task system as *SRT-feasible* if for any pattern of task invocations as allowed by the sporadic model, there exists a schedule that guarantees bounded tardiness for *every* task in the system. To illustrate, an overloaded system is clearly *not* SRT-feasible. A scheduling algorithm is *SRT-optimal* if, for *any* SRT-feasible system, every task is guaranteed bounded tardiness under that algorithm. Note that optimality is defined here with respect to schedulability rather than response times. That is, just as HRT-optimal schedulers (*e.g.*, EDF on uniprocessors) do not necessarily guarantee minimum response times, SRT-optimal ones also do not necessarily guarantee minimum deadline tardiness.

Devi and Anderson [2] showed that both the preemptive and non-preemptive GEDF schedulers are SRT-optimal for scheduling sporadic tasks on identical multiprocessors. In this paper, we consider the applicability of these results on heterogeneous multiprocessors platforms, which have been the subject of growing attention. As explained more carefully later, on a heterogeneous platform, processors can differ with respect to *speed* and/or *functionality*. If processors can only differ with respect to speed, then the platform is a *uniform* platform. Identical multiprocessors are a special case of uniform ones where all processors happen to have the same speed. In this paper, we examine and explore the SRT-optimality of GEDF scheduling when the underlying platform is extended from an identical one to a uniform heterogeneous one.

Related work. Research on EDF scheduling on uniform heterogeneous multiprocessors was initiated by Funk *et al.* They proposed two GEDF-based schedulers, f-EDF (full migrations) and r-EDF (restricted migrations). See Funk's dissertation [6] for details. However, their work only considered HRT systems, and therefore focused on approximation ratios and utilization bounds instead of optimality.

Devi and Anderson's original work on the SRT-optimality of preemptive and non-preemptive GEDF scheduling was later extended in several ways, *e.g.* by tightening the tardiness bounds [5], introducing the broader families of window-constrained schedulers [8] and GEDF-like (GEL) schedulers [4], and developing the global fair lateness scheduler [4], which is a GEL scheduler with better tardiness bounds than GEDF.

EDF-based SRT scheduling on uniform platforms was first considered by Leontyev and Anderson [9]. They presented a scheme wherein the platform is partitioned into

*Work supported by NSF grants CNS 1115284, CNS 1218693, CPS 1239135, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, and a grant from General Motors.

[1] See Sec. 2 for formal definitions.

"clusters" of identical processors, where the GEDF scheduler is used within each cluster, and some limited task migrations are allowed across clusters. Additionally, EDF-based SRT semi-partitioned [13] and global [11] schedulers have been proposed for uniform platforms. Although the algorithms proposed in each of the three just-cited papers are claimed to entail no *total* utilization loss—*i.e.*, the task set utilization can be as high as the platform's capacity—they all require some *per-task* utilization constraints, which compromise optimality. For example, the illustrative systems considered in Secs. 4 and 5 are excluded by the schedulability conditions in each of these papers, yet these systems are indeed feasible. In recent work, we showed that, if the sporadic task model is loosened by allowing intra-task parallelism—*i.e.*, consecutive invocations of the same task can proceed in parallel—then both the preemptive and non-preemptive GEDF schedulers are SRT-optimal on uniform platforms [12]. We have also proposed a semi-partitioned scheduler that is optimal (either HRT or SRT, depending on a configuration parameter) when scheduling conventional sporadic tasks on such platforms [14].

Contributions. We study the GEDF SRT-optimality results of Devi and Anderson [2] when the platform is extended to a uniform one. Their results show that, on identical multiprocessors, both the preemptive and non-preemptive GEDF schedulers are SRT-optimal for an arbitrary number of processors. On uniform multiprocessors, we prove that the preemptive GEDF scheduler is SRT-optimal on platforms of only two processors. Unfortunately, the proof techniques inherited from [2] do not easily extend to uniform platforms of three or more processors, and we explain why. Thus, we leave the optimality of preemptive GEDF scheduling on such platforms as an open problem. In the case of non-preemptive scheduling, we show that no *work-conserving* scheduler, including GEDF, can be SRT-optimal on uniform platforms, even if the number of processors is limited to two. Finally, inspired by these negative results, we suggest several avenues for future work. To the best of our knowledge, this is the first paper to consider the SRT-optimality of GEDF on uniform heterogeneous multiprocessors.

Organization. In the rest of this paper, we provide needed background (Sec. 2), derive a tardiness bound for the preemptive GEDF scheduler on uniform platforms of two processors (Sec. 3), discuss difficulties that arise when seeking a similar result for platforms of more than two processors (Sec. 4), show that no work-conserving non-preemptive scheduler can be SRT-optimal on uniform platforms (Sec. 5), and conclude (Sec. 6).

2 Background

According to [6, 10], multiprocessor platforms can be classified among three heterogeneity levels according to assumptions about processor speeds—the *speed* of a processor refers to the amount of work completed in one time unit when a task is executed on that processor.

- **Identical multiprocessors.** Every task is executed on any processor at the same speed, which is usually normalized to be 1.0 for simplicity.

- **Uniform heterogeneous multiprocessors.** Different processors may have different speeds, but on a given processor, every task is executed at the same speed. The speed of processor p is denoted s_p.

- **Unrelated heterogeneous multiprocessors.** The execution speed of a task depends on both the processor on which it is executed and the task itself, *i.e.*, a given processor may execute different tasks at different speeds. The execution speed of task τ_i on processor p is denoted $s_{p,i}$.

We specify a sporadic task τ_i by (C_i, T_i), where C_i is its *worst-case execution requirement*, which is defined as its worst-case execution time on a unit-speed processor, and T_i is its *period*, or minimum separation between any two consecutive invocations of the task. We also assume that the deadlines are *implicit* (*i.e.*, the relative deadline of each task equals its period) and each individual task is *sequential* (*i.e.*, intra-task parallelism is not allowed and each invocation of a task cannot commence execution until all previous invocations of that task complete). τ_i's *utilization* is defined as

$$u_i = \frac{C_i}{T_i}. \tag{1}$$

Note that, on a heterogeneous platform, $u_i \leq 1$ does not necessarily hold. Needed restrictions on utilizations, which depend on processor speeds, are given later. We consider the problem of scheduling a task set τ of n, where $n \geq 2$, such tasks on π, *i.e.*, $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. We define $C_{max} = \max_{1 \leq i \leq n}\{C_i\}$, and denote the total utilization of the task set τ by $U_\tau = \sum_{i=1}^{n} u_i$. Also, we assume that time is continuous.

A *job* is an invocation of a task. The j^{th} job of τ_i is denoted as $\tau_{i,j}$ and is released at time $r_{i,j}$ with an absolute deadline $d_{i,j} = r_{i,j} + T_i$. The time interval $[r_{i,j}, d_{i,j}]$ is called the *scheduling window* of $\tau_{i,j}$. A job is *ready* if and only if it is released and all previous jobs of the same task have already completed. If $\tau_{i,j}$ completes at time t_c, then its *tardiness* is defined as $\max\{0, t_c - d_{i,j}\}$. The tardiness of a *task* is the maximum tardiness of any of its jobs.

In [7], Funk *et al.* proved that a set of n implicit-deadline *periodic* tasks is HRT-feasible on an m-processor uniform platform with speeds $\{s_i\}$, where $1 \leq i \leq m$, if and only if

$$U_\tau \leq \sum_{i=1}^{m} s_i, \tag{2}$$

and

$$\sum_{k \text{ largest}} u_i \leq \sum_{k \text{ largest}} s_i \quad \text{for } k = 1, 2, \ldots, m-1. \tag{3}$$

In fact, the proof in [7] also shows that (2) and (3) are a necessary and sufficient feasibility condition (HRT or SRT) for implicit-deadline *sporadic* task systems.

Task set instantiation. Since the sporadic task model only requires a *minimum* job separation, there are an infinite number of *instantiations*, or job release patterns, of a certain sporadic task, as well as of a certain sporadic task set. We use $\mathcal{T}$ denote an instantiation of the sporadic task set τ.

Ideal schedule. We let $\pi_{\mathcal{I}} = \{u_1, u_2, \ldots, u_n\}$ denote an ideal multiprocessor for the task set τ, where $\pi_{\mathcal{I}}$ consists of n processors with speeds that exactly match the utilizations of the n tasks in τ, respectively. Let $\mathcal{I}$ be the partitioned schedule for τ on $\pi_{\mathcal{I}}$, where each task τ_i in τ is assigned to the processor of speed u_i. Then, in $\mathcal{I}$, every job in τ commences execution at its release time and completes execution within one period (it exactly executes for one period if and only if its actual execution requirement matches its worst-case execution requirement). Thus, all deadlines are met in $\mathcal{I}$. Also, note that, in $\mathcal{I}$, every job only executes within its scheduling window.

Definition of lag. Let $\mathsf{A}(\mathcal{T}, \mathcal{S}, \tau_i, t_1, t_2)$ denote the cumulative processor capacity allocation to task τ_i in an arbitrary schedule $\mathcal{S}$ of the instantiation $\mathcal{T}$ within the time interval $[t_1, t_2]$. By the definition of the ideal schedule $\mathcal{I}$, for any $\mathcal{T}$,

$$\mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, t_1, t_2) \leq u_i \cdot (t_2 - t_1). \tag{4}$$

For an arbitrary schedule $\mathcal{S}$ of an arbitrary instantiation $\mathcal{T}$, we denote the difference between the allocation to a task τ_i in $\mathcal{I}$ and in $\mathcal{S}$ within $[0, t)$ as

$$\mathsf{lag}(\mathcal{T}, \tau_i, t, \mathcal{S}) = \mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, 0, t) - \mathsf{A}(\mathcal{T}, \mathcal{S}, \tau_i, 0, t). \tag{5}$$

Thus, we have

$$\begin{aligned} \mathsf{lag}(\mathcal{T}, \tau_i, t_2, \mathcal{S}) = {} & \mathsf{lag}(\mathcal{T}, \tau_i, t_1, \mathcal{S}) + \\ & \mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, t_1, t_2) - \mathsf{A}(\mathcal{T}, \mathcal{S}, \tau_i, t_1, t_2). \end{aligned} \tag{6}$$

Also, we define

$$\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) = \sum_{\tau_i \in \tau} \mathsf{lag}(\mathcal{T}, \tau_i, t, \mathcal{S}). \tag{7}$$

3 Preemptive GEDF on Two Processors

In this section, we prove that under preemptive GEDF scheduling, for any feasible task set on a two-processor uniform platform, every job has a bounded tardiness, *i.e.*, preemptive GEDF is SRT-optimal on dual-core uniform heterogeneous multiprocessors.

Considered platform. We consider a uniform dual-core multiprocessor π, where the higher-speed (lower-speed) core is denoted by its speed s_h (s_l), *i.e.*, $\pi = \{s_h, s_l\}$ where $s_h \geq s_l$. By (2) and (3), the feasibility conditions on the dual-core uniform platform π reduce to

$$U_\tau \leq s_h + s_l, \tag{8}$$

and

$$u_i \leq s_h \quad \text{for any } i. \tag{9}$$

Preemptive GEDF. The preemptive GEDF scheduling algorithm on π is defined as follows. For any time instant, the ready but incomplete job with the earliest deadline (the second earliest deadline) is scheduled on s_h (s_l) if there is at least one (two) ready but incomplete job(s). Deadline ties are broken arbitrarily. In the rest of this section, we use $\mathcal{S}$ to denote the preemptive GEDF schedule for an arbitrary instantiation $\mathcal{T}$ of a sporadic task set τ that is feasible on π.

A key property of preemptive GEDF scheduling is that, for any job, its execution does not depend on any jobs with a later deadline. We will use this property multiple times in the proof in this section.

We define the deadline of a task as the deadline of its most recently ready job, *i.e.*, the deadline of a task τ_i at time instant t is defined as $d_i(t) = d_{i,j}$, where $\tau_{i,j}$ is the most recently ready job of τ_i. Note that the most recently *ready* job is not necessary to be the most recently *released* job. For example, if $\tau_{i,j+1}$ is released at time t, but $\tau_{i,j}$ has not completed by time t, then $\tau_{i,j+1}$ is not considered to be *ready* at time t. Then, assuming $\tau_{i,j}$ is ready at time t, the most recently ready job of τ_i at time t should be $\tau_{i,j}$ rather than $\tau_{i,j+1}$.

If $U_\tau \leq s_h$, then it is clear that every job will meet its deadline and therefore has zero tardiness. This is because by the optimality of EDF on uniprocessors, all deadlines will be met if we only schedule jobs on s_h. Furthermore, compared to this uniprocessor schedule, the GEDF scheduling described above does not *delay* any execution of any job. Thus, in the rest of this section, we focus on the other case, *i.e.*, $s_h < U_\tau \leq s_h + s_l$.

We define a time instant as *busy* if and only if at that time instant both s_h and s_l are executing a job, and *non-busy* otherwise. We define a time interval to be a *busy interval* (*non-busy interval*) if and only if every time instant in this interval is a busy (non-busy) time instant. A busy interval $(t_1, t_2]$ has the following property.

Lemma 1. *For any busy interval* $(t_1, t_2]$,

$$\mathsf{LAG}(\mathcal{T}, \tau, t_1, \mathcal{S}) \geq \mathsf{LAG}(\mathcal{T}, \tau, t_2, \mathcal{S}).$$

Proof. Since $(t_1, t_2]$ is a busy interval,

$$\sum_{\tau_i \in \tau} \mathsf{A}(\mathcal{T}, \mathcal{S}, \tau_i, t_1, t_2) = (s_h + s_l) \cdot (t_2 - t_1). \tag{10}$$

By (4),

$$\sum_{\tau_i \in \tau} \mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, t_1, t_2) \leq \sum_{\tau_i \in \tau} u_i \cdot (t_2 - t_1) = U_\tau \cdot (t_2 - t_1). \tag{11}$$

Thus,

$$\begin{aligned}
&\mathsf{LAG}(\mathcal{T}, \tau, t_2, \mathcal{S}) \\
&= \{\text{by (7)}\} \\
&\quad \sum_{\tau_i \in \tau} \mathsf{lag}(\mathcal{T}, \tau_i, t_2, \mathcal{S}) \\
&= \{\text{by (6)}\} \\
&\quad \sum_{\tau_i \in \tau} \Big(\mathsf{lag}(\mathcal{T}, \tau_i, t_1, \mathcal{S}) + \\
&\qquad \mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, t_1, t_2) - \mathsf{A}(\mathcal{T}, \mathcal{S}, \tau_i, t_1, t_2) \Big) \\
&= \{\text{by (7)}\} \\
&\quad \mathsf{LAG}(\mathcal{T}, \tau, t_1, \mathcal{S}) + \sum_{\tau_i \in \tau} \mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, t_1, t_2) - \\
&\qquad \sum_{\tau_i \in \tau} \mathsf{A}(\mathcal{T}, \mathcal{S}, \tau_i, t_1, t_2) \\
&\leq \{\text{by (10) and (11)}\} \\
&\quad \mathsf{LAG}(\mathcal{T}, \tau, t_1, \mathcal{S}) + \big(U_\tau - (s_h + s_l)\big) \cdot (t_2 - t_1) \\
&\leq \{\text{by (8) and } t_1 \leq t_2\} \\
&\quad \mathsf{LAG}(\mathcal{T}, \tau, t_1, \mathcal{S}).
\end{aligned}$$

The lemma follows. □

Lemma 2. *If at time instant t, both s_h and s_l are idle, then* $\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) \leq 0$.

Proof. If $\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) > 0$, then by (7), there exists a task τ_i such that $\mathsf{lag}(\mathcal{T}, \tau_i, t, \mathcal{S}) > 0$. By the definition of lag and the ideal schedule $\mathcal{I}$, this implies that, at time t, τ_i has a job that is ready and incomplete. Thus, if there is an idle processor, τ_i should be scheduled, which contradicts the assumption that both s_h and s_l are idle at time t. □

Lemma 3. *At time instant t, for any task τ_i, if $t \leq d_i(t)$, then* $\mathsf{lag}(\mathcal{T}, \tau_i, t, \mathcal{S}) \leq C_i$.

Proof. Suppose $\tau_{i,r}$ is the most recently ready job of τ_i in $\mathcal{S}$ at time t. Then $t \leq d_i(t) = d_{i,r}$. Let $e_{i,j}$ denote the *actual* execution requirement of $\tau_{i,j}$. Then by the definition of C_i, $e_{i,j} \leq C_i$ for any j. Since $\tau_{i,r}$ is ready in $\mathcal{S}$ at time t, $\tau_{i,r-1}$ must complete by t. That is,

$$\mathsf{A}(\mathcal{T}, \mathcal{S}, \tau_i, 0, t) \geq \sum_{j=1}^{r-1} e_{i,j}. \tag{12}$$

Also, by the definition of $\mathcal{I}$ and since $t \leq d_{i,r}$,

$$\mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, 0, t) \leq \sum_{j=1}^{r} e_{i,j}. \tag{13}$$

Thus, by (5),

$$\begin{aligned}
\mathsf{lag}(\mathcal{T}, \tau_i, t, \mathcal{S}) &\leq \sum_{j=1}^{r} e_{i,j} - \sum_{j=1}^{r-1} e_{i,j} \\
&= e_{i,r} \\
&\leq C_i.
\end{aligned}$$

The lemma follows. □

Lemma 4. *Suppose an arbitrary task τ_i has completed its most recently ready job at time instant t. Then,* $\mathsf{lag}(\mathcal{T}, \tau_i, t, \mathcal{S}) \leq 0$ *if* $t \leq d_i(t)$; $\mathsf{lag}(\mathcal{T}, \tau_i, t, \mathcal{S}) = 0$ *if* $t > d_i(t)$.

Proof. Suppose $\tau_{i,r}$ is the most recently ready job of τ_i in $\mathcal{S}$ at time t. Then $t \leq d_i(t) = d_{i,r}$. Let $e_{i,j}$ denote the *actual* execution requirement of $\tau_{i,j}$. Then by the definition of C_i, $e_{i,j} \leq C_i$ for any j.

Since τ_i has completed its most recently ready job $\tau_{i,r}$ by time t,

$$\mathsf{A}(\mathcal{T}, \mathcal{S}, \tau_i, 0, t) = \sum_{j=1}^{r} e_{i,j}. \tag{14}$$

Also, by the definition of $\mathcal{I}$,

$$\mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, 0, d_{i,r}) = \sum_{j=1}^{r} e_{i,j}. \tag{15}$$

Case 1: $t \leq d_{i,r}$**.** By the definition of $\mathcal{I}$,

$$\mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, 0, t) \leq \mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, 0, d_{i,r}). \tag{16}$$

By (5), (14), (15), and (16), $\mathsf{lag}(\mathcal{T}, \tau_i, t, \mathcal{S}) \leq 0$.

Case 2: $t > d_{i,r}$**.** Since $\tau_{i,r}$ is its the most recently ready job, τ_i does not release any job in the time interval $[d_{i,r}, t]$. Thus, by the definition of $\mathcal{I}$, $\mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, d_{i,r}, t) = 0$, *i.e.*,

$$\mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, 0, t) = \mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, 0, d_{i,r}). \tag{17}$$

By (5), (14), (15), and (17), $\mathsf{lag}(\mathcal{T}, \tau_i, t, \mathcal{S}) = 0$.

By combining Cases 1 and 2, the lemma follows. □

Lemma 5. *For any $t \geq 0$, for any instantiation $\mathcal{T}$ of the sporadic task set τ,* $\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S})$ *is at most C_{max}.*

Proof. We prove this lemma by induction.

Base case. It is clear that, when $t = 0$, for any instantiation $\mathcal{T}$ of the sporadic task set τ, $\mathsf{LAG}(\mathcal{T}, \tau, 0, \mathcal{S}) = 0$. Thus, the lemma follows for $t = 0$.

Inductive step. For an arbitrary time instant $t > 0$, suppose that for any time instant $t' < t$ the lemma follows. We prove the lemma follows for t as well.

Case 1: t is a busy instant. Let t_n denote the latest non-busy time instant before t if such an time instant exists, otherwise, let $t_n = 0$. Then, $t_n < t$ and $(t_n, t]$ is a busy interval. By Lem. 1,

$$\mathsf{LAG}(\mathcal{T}, \tau, t_n, \mathcal{S}) \geq \mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) \tag{18}$$

Since $t_n < t$, by the inductive assumption,

$$\mathsf{LAG}(\mathcal{T}, \tau, t_n, \mathcal{S}) \leq C_{max} \tag{19}$$

By (18) and (19), $\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) \leq C_{max}$.

Case 2: t is a non-busy instant. If both s_h and s_l are idle at time t, then by Lem. 2, $\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) \leq 0 < C_{max}$. Since t is non-busy and the preemptive GEDF scheduler prefers s_h over s_l, the remaining possibility to consider is that, at time t, some task τ_k executes on s_h and s_l is idle. Note that, any task other than τ_k must have completed its most recently ready job at time t; otherwise, such task would also be scheduled at time t, which contradicts the assumption that t is non-busy. Thus, by Lem. 4,

$$\mathsf{lag}(\mathcal{T}, \tau_i, t, \mathcal{S}) \leq 0, \quad \text{for any } i \neq k. \tag{20}$$

Let $\tau_{k,r}$ denote the most recently ready job of τ_k at time t, *i.e.*, $d_k(t) = d_{k,r}$. Also, let $t_d = d_k(t) = d_{k,r}$ for notational simplicity.

Case 2.1: $t \leq t_d$. By Lem. 3, $\mathsf{lag}(\mathcal{T}, \tau_k, t, \mathcal{S}) \leq C_k \leq C_{max}$. Thus, by (20) and (7), $\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) \leq C_{max}$.

Case 2.2: $t > t_d$. In this case, we consider an instantiation $\mathcal{T}'$, which is identical to $\mathcal{T}$ except that, all jobs of tasks other than τ_k that have a deadline after t_d and have completed before t are *dropped* (or equivalently, are never released). Fig.1(a) is an illustrates this situation. Note that, by the definition of sporadic tasks, $\mathcal{T}'$ is also an instantiation of the sporadic task set τ. Let $\mathcal{S}'$ denote the preemptive GEDF schedule of $\mathcal{T}'$. Thus, by the inductive assumption, applying this lemma to $\mathcal{T}'$ (since the lemma is with respect to *any* instantiation of τ),

$$\mathsf{LAG}(\mathcal{T}', \tau, t', \mathcal{S}') \leq C_{max}, \quad \text{for any } t' < t. \tag{21}$$

In $\mathcal{T}'$, compared with $\mathcal{T}$, we did not change the releases of τ_k, and under the GEDF scheduling, τ_k's execution until t only depends on the execution of jobs with a deadline at or before t_d. The set of such jobs is not impacted by dropping jobs with a deadline after t_d. Therefore, until t, the scheduling of τ_k is identical in both $\mathcal{T}$ and $\mathcal{T}'$, *i.e.*,

$$\mathsf{lag}(\mathcal{T}, \tau_k, t, \mathcal{S}) = \mathsf{lag}(\mathcal{T}', \tau_k, t, \mathcal{S}'). \tag{22}$$

Also, for any task τ_i other than τ_k, it is clear that such job dropping does not change the fact that τ_i has completed its most recently ready job at time t (note that, in $\mathcal{T}$ or $\mathcal{T}'$, "most recently ready job at time t" may refer to different jobs, since in $\mathcal{T}'$ some jobs in $\mathcal{T}$ may be never released). The only impact to τ_i is that, in $\mathcal{T}'$, $d_i(t) \leq t_d$ for any $i \neq k$. Since in this subcase (Case 2.2) $t > t_d$, we have $d_i(t) < t$ for any $i \neq k$ as well. Thus, by Lem. 4,

$$\mathsf{lag}(\mathcal{T}', \tau_i, t, \mathcal{S}') = 0, \quad \text{for any } i \neq k, \tag{23}$$

which implies, by (20) and (23),

$$\mathsf{lag}(\mathcal{T}, \tau_i, t, \mathcal{S}) \leq \mathsf{lag}(\mathcal{T}', \tau_i, t, \mathcal{S}'), \quad \text{for any } i \neq k. \tag{24}$$

Thus, by (7), (22), and (24), we have

$$\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) \leq \mathsf{LAG}(\mathcal{T}', \tau, t, \mathcal{S}'). \tag{25}$$

That is, it is sufficient to show $\mathsf{LAG}(\mathcal{T}', \tau, t, \mathcal{S}') \leq C_{max}$. We first prove the following preliminary Claim.

Claim 1. *If a task τ_k is continuously executing throughout a non-busy interval $(t_1, t_2]$ and no any other task has a job with a scheduling window that overlaps with $(t_1, t_2]$, then for any instantiation $\mathcal{T}$, $\mathsf{LAG}(\mathcal{T}, \tau, t_1, \mathcal{S}) \geq \mathsf{LAG}(\mathcal{T}, \tau, t_2, \mathcal{S})$.*

Proof. Since τ_i is continuously executing throughout the time interval $(t_1, t_2]$,

$$\mathsf{A}(\mathcal{T}, \mathcal{S}, \tau_k, t_1, t_2) = s_h \cdot (t_2 - t_1), \tag{26}$$

and since any time instant within $[t_1, t_2]$ is non-busy,

$$\mathsf{A}(\mathcal{T}, \mathcal{S}, \tau_i, t_1, t_2) = 0, \quad \text{for any } i \neq k. \tag{27}$$

Since no any other task has a job with a scheduling window that overlaps with $[t_1, t_2]$,

$$\mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_i, t_1, t_2) = 0, \quad \text{for any } i \neq k. \tag{28}$$

Also, by (4),

$$\mathsf{A}(\mathcal{T}, \mathcal{I}, \tau_k, t_1, t_2) \leq u_k \cdot (t_2 - t_1), \tag{29}$$

By (6), (7), (26), (27), (28), and (29),

$$\begin{aligned}\mathsf{LAG}(\mathcal{T}, \tau, t_2, \mathcal{S}) = {} & \mathsf{LAG}(\mathcal{T}, \tau, t_1, \mathcal{S}) \\ & + (u_k - s_h) \cdot (t_2 - t_1) \\ \leq {} & \{\text{by (9) and since } t_1 \leq t_2\} \\ & \mathsf{LAG}(\mathcal{T}, \tau, t_1, \mathcal{S}).\end{aligned}$$

The claim follows. □

Let t_b denote the latest busy time instant before time t if such an instant exists, otherwise, let $t_b = 0$. t_b is well-defined such that $t_b < t$, because by the condition of Case 2, t is non-busy.

Case 2.2.1: $t_b \leq t_d$. That is, $(t_d, t]$ is a non-busy interval, and therefore τ_k is continuously executing on s_h within this non-busy interval, as shown in Fig.1(b). Moreover, in $\mathcal{T}'$, $d_i(t) \leq t_d$ for any $i \neq k$, so no task other than τ_k has a job with a scheduling window that overlaps with $(t_d, t]$. Thus, by Claim 1,

$$\mathsf{LAG}(\mathcal{T}', \tau, t, \mathcal{S}') \leq \mathsf{LAG}(\mathcal{T}', \tau, t_d, \mathcal{S}'). \tag{30}$$

Since $t_d < t$ (the condition of Case 2.2), by (21),

$$\mathsf{LAG}(\mathcal{T}', \tau, t_d, \mathcal{S}') \leq C_{max}. \tag{31}$$

Thus, by (25), (30),and (31), $\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) \leq C_{max}$.

Case 2.2.2: $t_b > t_d$. That is, $(t_b, t]$ is a non-busy interval, and therefore τ_k is continuously executing on s_h within this non-busy interval, as shown in Fig.1(c). Moreover, in $\mathcal{T}'$, $d_i(t) \leq t_d \leq t_b$ for any $i \neq k$, so no task other than τ_k has a job with a scheduling window that overlaps with $(t_b, t_n]$. Thus, by Claim 1,

$$\mathsf{LAG}(\mathcal{T}', \tau, t, \mathcal{S}') \leq \mathsf{LAG}(\mathcal{T}', \tau, t_b, \mathcal{S}'). \tag{32}$$

Since $t_b < t$ (by the definition of t_b), by (21),

$$\mathsf{LAG}(\mathcal{T}', \tau, t_b, \mathcal{S}') \leq C_{max}. \tag{33}$$

Thus, by (25), (32),and (33), $\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) \leq C_{max}$.

Conclusion. The cases above are exhaustive, and in all cases, we have $\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) \leq C_{max}$. That is, we have proved the inductive step, *i.e.*, for a certain time instant $t > 0$, assuming that $\mathsf{LAG}(\mathcal{T}, \tau, t', \mathcal{S}) \leq C_{max}$ for any time instant $t' < t$ for any instantiation of τ, we have proved that $\mathsf{LAG}(\mathcal{T}, \tau, t, \mathcal{S}) \leq C_{max}$ for any instantiation of τ. Thus, combining the base case at the beginning of the proof, the lemma follows by induction. □

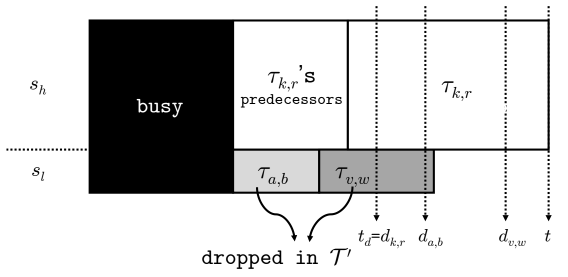
Theorem 1. *Under preemptive GEDF scheduling, for any feasible sporadic task set τ on π, every job has a tardiness at most C_{max}/s_h.*

Proof. Let $\tau_{i,j}$ denote an arbitrary job in an arbitrary instantiation $\mathcal{T}$ of τ, and let t_d denote the absolute deadline of $\tau_{i,j}$, *i.e.*, $t_d = d_{i,j}$. We prove that $\tau_{i,j}$ completes its execution by $t_d + C_{max}/s_h$.

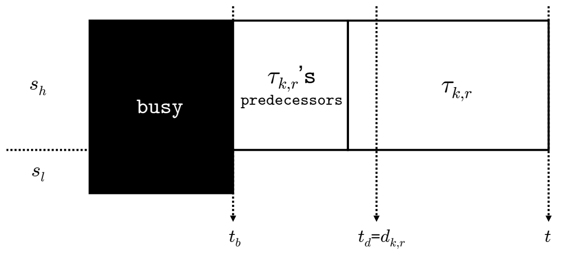
For any instantiation of τ, since no job with a deadline at or before t_d could be released after t_d, the following property holds for any instantiation of τ under preemptive GEDF scheduling.

(P1) Once $\tau_{i,j}$ is scheduled on s_h at or after t_d, it continuously executes on s_h until it completes.

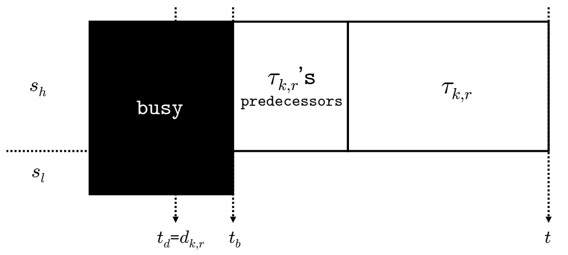
Also, for any instantiation $\mathcal{T}$, we can consider an instantiation $\mathcal{T}'$, which is identical to $\mathcal{T}$ except that, all jobs with a deadline after t_d are *dropped* (or equivalently, never released). Note that $\mathcal{T}'$ is also a valid instantiation of the sporadic task set τ. Also, let $\mathcal{S}'$ denote the preemptive GEDF schedule for $\mathcal{T}'$ on π. Then, the following claim holds.



(a) Illustration for Case 2.2: constructing $\mathcal{T}'$ and $\mathcal{S}'$ from $\mathcal{S}$.



(b) Illustration for Case 2.2.1: schedule $\mathcal{S}'$.



(c) Illustration for Case 2.2.2: schedule $\mathcal{S}'$.

Figure 1: Illustrative schedules for Lem. 5.

Claim 2. *The tardiness of $\tau_{i,j}$ in $\mathcal{S}'$ is identical to that in $\mathcal{S}$.*

Proof. In both $\mathcal{S}$ and $\mathcal{S}'$, $\tau_{i,j}$'s execution only depends on the execution of jobs with a deadline at or before t_d, so its execution is not impacted by dropping jobs with a deadline after t_d. Therefore, the claim follows. □

By Claim 2, in the rest of this proof, it is sufficient to consider the instantiation $\mathcal{T}'$ instead, and to upper bound $\tau_{i,j}$'s tardiness in $\mathcal{S}'$. Since in $\mathcal{T}'$ all jobs with a deadline after t_d are dropped, we have the following properties for $\mathcal{T}'$.

(P2) After t_d, once π becomes non-busy, it will be continually non-busy afterwards.

(P3) At t_d, the remaining work for $\mathcal{T}'$ (if any), including the remaining work for $\tau_{i,j}$, is upper bounded by $\mathsf{LAG}(\mathcal{T}', \tau, t_d, \mathcal{S}')$.

We inductively assume $\tau_{i,j}$'s predecessor jobs have tardiness at most C_{max}/s_h, and then prove $\tau_{i,j}$ has tardiness at most C_{max}/s_h as well. For $\tau_{i,1}$, which has no actual predecessor, we define a "virtual predecessor" $\tau_{i,0}$, which has a deadline at time zero and zero tardiness and which yields the base case of the induction. This enables us to avoid distracting boundary cases.

By this inductive assumption, any jobs with a deadline earlier than t_d have tardiness at most C_{max}/s_h, and by the definition of the period of a task, $\tau_{i,j-1}$ has a deadline at most $t_d - T_i$. Therefore, $\tau_{i,j-1}$ must complete by $t_d - T_i + C_{max}/s_h$. Moreover, $\tau_{i,j}$ is released at $t_d - T_i$. Thus, the following property holds.

(P4) $\tau_{i,j}$ is ready by $t_d - T_i + C_{max}/s_h$.

Let t_b denote the latest busy time instant after t_d if such an instant exists, otherwise, let $t_b = t_d$. Then, by **(P2)**, $(t_d, t_b]$ must be a busy interval, in which the amount of work that is done is $(s_h + s_l) \cdot (t_b - t_d)$.

Case 1: $t_b \leq t_d - T_i + C_{max}/s_h$. In this case, by **(P1)**, **(P2)**, and **(P4)**, $\tau_{i,j}$ either has completed by time $t_d - T_i + C_{max}/s_h$, or continually executes on s_h after time $t_d - T_i + C_{max}/s_h$ and thus must complete within C_i/s_h time units. That is, $\tau_{i,j}$ must complete by time $t_d - T_i + C_{max}/s_h + C_i/s_h$. By (1) and (9), $C_i/s_h \leq T_i$. Thus, $\tau_{i,j}$ must complete by time $t_d + C_{max}/s_h$.

Case 2: $t_b > t_d - T_i + C_{max}/s_h$. In this case, **(P4)** implies that $\tau_{i,j}$ is ready by time t_b as well. Then, by **(P1)** and **(P2)**, $\tau_{i,j}$ either has completed by time t_b, or continually executes on s_h after time t_b. Let δ denote the remaining execution requirement of $\tau_{i,j}$ that has not completed at time t_b. Then, $\tau_{i,j}$ must complete within δ/s_h time units, *i.e.*, by time $t_b + \delta/s_h$. Moreover, no task other than τ_i would have remaining work after time t_b, otherwise t_b would not be the latest busy time instant. Therefore, by **(P3)**, we have $(s_h + s_l) \cdot (t_b - t_d) + \delta \leq \mathsf{LAG}(\mathcal{T}', \tau, t_d, \mathcal{S}')$. That is, $t_b \leq t_d + \big(\mathsf{LAG}(\mathcal{T}', \tau, t_d, \mathcal{S}') - \delta\big)/(s_h + s_l)$. Thus,

$$\begin{aligned}
t_b + \frac{\delta}{s_h} &\leq t_d + \frac{\mathsf{LAG}(\mathcal{T}', \tau, t_d, \mathcal{S}') - \delta}{s_h + s_l} + \frac{\delta}{s_h}\\
&= \{\text{rearranging}\}\\
&\quad t_d + \frac{\mathsf{LAG}(\mathcal{T}', \tau, t_d, \mathcal{S}')}{s_h + s_l} + \delta \cdot \left(\frac{1}{s_h} - \frac{1}{s_h + s_l}\right)\\
&\leq \{\text{by Lem. 5}\}\\
&\quad t_d + \frac{C_{max}}{s_h + s_l} + \delta \cdot \left(\frac{1}{s_h} - \frac{1}{s_h + s_l}\right)\\
&\leq \{\text{since } \delta \leq C_i\}\\
&\quad t_d + \frac{C_{max}}{s_h + s_l} + C_i \cdot \left(\frac{1}{s_h} - \frac{1}{s_h + s_l}\right)\\
&\leq \{\text{since } C_i \leq C_{max}, \text{ and simplifying}\}\\
&\quad t_d + \frac{C_{max}}{s_h}.
\end{aligned}$$

That is, $\tau_{i,j}$ completes by $t_d + C_{max}/s_h$.

Combining Cases 1 and 2, we have proved that an arbitrary job $\tau_{i,j}$ has a tardiness at most C_{max}/s_h. Thus, the theorem follows. □

Corollary 1. Under preemptive GEDF scheduling on two identical processors, every job has a tardiness at most C_{max}.

Proof. By setting $s_h = s_l = 1.0$, the corollary follows from Thm. 1 □

Corollary 1 exactly matches a result proven in [2] for two processors.

4 Preemptive GEDF on m Processors

In Sec. 3, we proved that the preemptive GEDF scheduler is SRT-optimal on two-processor uniform platforms. Unfortunately, the proof techniques inherited from [2] are unlikely to apply to larger uniform platforms. This is because the situation for tardy tasks in a non-busy interval has changed.

The main technique in both Sec. 3 and [2] is to consider the schedule by busy intervals and non-busy intervals. The intuition behind this technique is that, throughout a busy interval, the set of all tasks, or the total system, is not becoming more tardy, whereas throughout a non-busy interval, any tardy task is not becoming more tardy. Moreover, any schedule can be split into sub-intervals that are either busy or non-busy.

Such intuition for busy intervals holds for uniform platform with an arbitrary number of processors, since, to be feasible, the system should not be overutilized. However, the intuition for non-busy intervals depends on the number of processors for uniform platforms. Let us say that a processor is *fast enough* for a task if the speed of the processor is at least the utilization of the task. Then, a task scheduled on a fast enough processor will not become more tardy. For any uniform platform, including identical multiprocessors, the following property holds: at any non-busy time instant, every tardy task is scheduled; otherwise, the scheduler is not work-conserving, which is a property of GEDF. On identical multiprocessors, every processor is fast enough for any task in a feasible system. Therefore, at any non-busy time instant, as every tardy task is scheduled, it is also not becoming more tardy. On two-processor uniform platforms, the faster processor is fast enough for any task in a feasible system, and a non-busy time instant implies that at that time instant at most one task is tardy, which should be scheduled on the faster processor and therefore is not becoming more tardy. Thus, the intuition for non-busy intervals holds for identical multiprocessors also holds for two-processor uniform platforms.

Nevertheless, this intuition for non-busy intervals may not hold for larger uniform platforms. Consider a three-processor uniform platform $\pi = \{5, 2, 2\}$ on which three tasks with utilizations $u_1 = u_2 = u_3 = 3$ are to be scheduled. By (2) and (3), this system is feasible. However, it could be the case that at a non-busy time instant, a tardy task

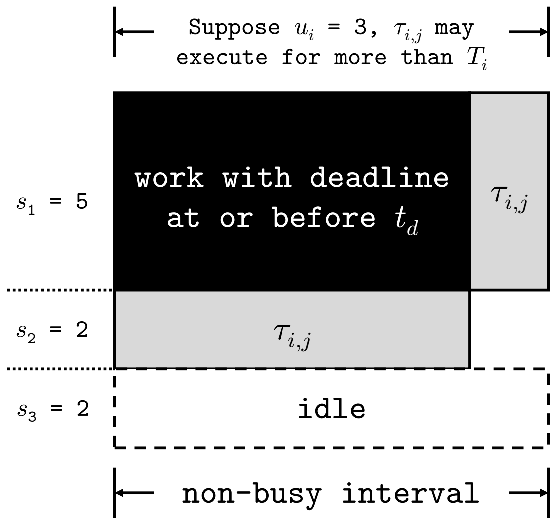

Suppose u_i = 3, $\tau_{i,j}$ may execute for more than T_i
s_1 = 5
work with deadline at or before t_d
$\tau_{i,j}$
s_2 = 2
$\tau_{i,j}$
s_3 = 2
idle
non-busy interval

Figure 2: On a uniform platform of more than three processors, a job of a task could continuously execute for more than its period within a non-busy interval.

is scheduled on a processor with speed of 2 because another more tardy one is scheduled on the processor with speed 5. Thus, at a non-busy time instant, a tardy task could be scheduled on a processor that is not fast enough, and hence the tardy task could become more tardy.

Technically, the impact of this difference is that Thm. 1, or more specifically Case 1 in the proof of Thm. 1, does not hold anymore. That part of the proof relies on the condition $C_i/s_h \leq T_i$, which actually expresses that s_h is fast enough for any task. However, on larger uniform platforms, the equivalent condition becomes $C_i/s_p \leq T_i$, where s_p could be any processor other than the slowest one (since a non-busy interval is under consideration). It is not guaranteed that s_p is fast enough for an arbitrary task τ_i. That is, if a job continuously executes in a non-busy interval, then it is no longer nessarily the case that this job will complete within one period, since it could be scheduled on a processor with a speed that is less than the utilization of the corresponding task. Fig. 2 illustrates this situation.

Of course, we could constrain the per-task utilizations to satisfy some conditions to ensure every task must execute on a fast enough processor, as [9] [13] [11] did. However, such restrictions would compromise optimality.

In conclusion, establishing the optimality of preemptive GEDF on more than three uniform processors is unlikely using the techniques in Sec. 3 and [2]. Unfortunately, we could not devise a counterexample to show that preemptive GEDF is not SRT-optimal on more than two uniform processors. Thus, whether preemptive GEDF is SRT-optimal on larger uniform platforms remains as an open problem.

5 Non-Preemptive Scheduling

Devi and Anderson [2] proved that both the preemptive and non-preemptive GEDF schedulers are SRT-optimal on identical multiprocessors. In Sec. 3, we proved that the preemptive GEDF scheduler is SRT-optimal on two-processor uniform platforms, and the proof is consistent with the proof of the SRT-optimality of GEDF on identical multiprocessors in [2]. Thus, we might intuitively speculate that the non-preemptive GEDF scheduler is SRT-optimal, at least on two uniform processors. Unfortunately, this is not true.

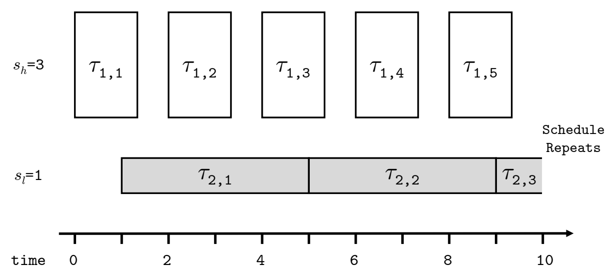
A non-preemptive scheduler schedules ready jobs, and once a job is scheduled, it continually executes without preemption until it completes. In this paper, we further require that under non-preemptive scheduling, once a job is scheduled, it continually executes *on the processor on which it was scheduled* without preemption until it completes, *i.e.*, non-preemptivity means no preemption and no *migration* occurs within a single job. This implicitly holds for any non-preemptive scheduler on identical multiprocessors, where every processor is the same and therefore there is no point to migrating a job that is currently executing; however, we do have to clarify this here, since the processors in a uniform platform could be of different speeds.

In this section, we prove that non-preemptive GEDF is not SRT-optimal on uniform platforms by proving that no *work-conserving* non-preemptive scheduler is SRT-optimal on uniform platforms by giving a counterexample. A *work-conserving* scheduler prevents the situation where there is at least one processor that is idle, and at least one task that has a incomplete ready job but is not scheduled, *i.e.*, whenever a task could be scheduled *somewhere*, it is scheduled. GEDF is clearly a work-conserving scheduler.

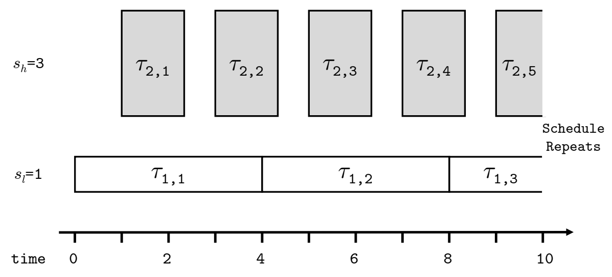
The counterexample. We consider a two-processor uniform platform $\pi = \{s_h = 3, s_l = 1\}$ and a task set of two tasks, $\tau_1 = (4, 2)$ and $\tau_2 = (4, 2)$, to be scheduled on π. Also, we consider the situation where τ_1 releases its first job at time 0 and then releases jobs as soon as possible, and τ_2 releases its first job at time 1 and then releases jobs as soon as possible. Furthermore, we assume every job has an execution requirement that matches its worst-case execution requirement.

At time 0, $\tau_{1,1}$ is released, a work-conserving scheduler must schedule it on either s_h or s_l.

Case 1: $\tau_{1,1}$ is scheduled on s_h (Fig. 3(a)). Then, $\tau_{1,1}$ continuously executes on s_h until time 1.33. Therefore, at time 1 when $\tau_{2,1}$ is released, s_l and only s_l is available. Thus, a work-conserving scheduler must schedule $\tau_{2,1}$ on s_l, where $\tau_{2,1}$ continuously executes until time 5, which means both $\tau_{1,2}$ (released at time 2) and $\tau_{1,3}$ (released at time 4) must be scheduled on s_h and each of them continuously executes on s_h for 1.33 time units. Thus, at time 5 when $\tau_{2,1}$ completes and $\tau_{2,2}$ is ready, s_l and only s_l is available, which means that a work-conserving scheduler must scheduler $\tau_{2,2}$ on s_l where $\tau_{2,2}$ continuously executes until time 9. This pattern



(a) Case 1.



(b) Case 2.

Figure 3: Counterexample schedules.

repeats in the schedule. Fig. 3(a) shows the schedule. Observe that τ_2 is always scheduled on s_l and therefore becomes unboundedly tardy.

Case 2: $\tau_{1,1}$ is scheduled on s_l (Fig. 3(b)). Then, $\tau_{1,1}$ continuously executes on s_h until time 4, which means both $\tau_{2,1}$ (released at time 1) and $\tau_{2,2}$ (released at time 3) must be scheduled on s_h and each of them continuously executes on s_h for 1.33 time units. Thus, at time 4 when $\tau_{1,1}$ completes and $\tau_{1,2}$ is ready, s_l and only s_l is available, which means that a work-conserving scheduler must scheduler $\tau_{1,2}$ on s_l where $\tau_{1,2}$ continuously executes until time 8. This pattern repeats in the schedule. Fig. 3(b) shows the schedule. Observe that τ_1 is always scheduled on s_l and therefore becomes unboundedly tardy.

Thus, in this system, under any work-conserving non-preemptive scheduler, there must be one task that has unbounded tardiness. However, by (2) and (3), this system is actually feasible. Fig. 4 shows a feasible schedule for this system where all deadlines are met.

In this counterexample, each task releases subsequent jobs as soon as possible, so it is valid not only for sporadic tasks, but also for periodic tasks where the two tasks have phases 0 and 1, respectively. Also, the two-processor uniform platform considered in this section is a special case for the more general uniform platform where the number of processors is arbitrary. Thus, the following theorem holds.

Theorem 2. *No work-conserving non-preemptive scheduler is SRT-optimal for sequential sporadic or periodic tasks on uniform heterogeneous multiprocessors, even if the number of processors is restricted to two.*

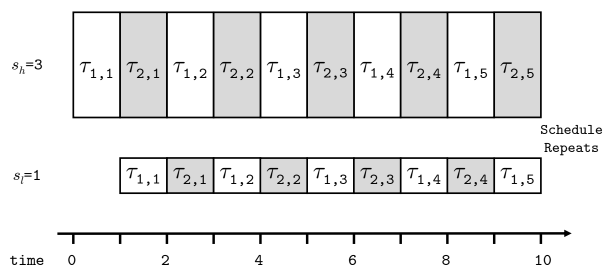


Figure 4: Feasible schedule.

One might wonder whether if this system with this job release pattern is SRT-feasible under the non-preemptive restriction, *i.e.*, under non-preemptive scheduling, whether it is possible to have bounded tardiness for every task in this system. In fact, this system with this job release pattern is indeed SRT-feasible for non-preemptive scheduling. For example, Fig. 5 is a non-preemptive schedule for this system, and deadline tardiness is at most 3 time units.

The key in the schedule in Fig. 5 is that it is not work-conserving. At time 4, $\tau_{1,3}$ is ready and s_h is idle, but in this schedule $\tau_{1,3}$ is not scheduled until time 5. At time 5, when $\tau_{2,1}$ has completed on s_l, we schedule $\tau_{2,2}$ on s_h, and schedule $\tau_{1,3}$ on s_l. Then, the schedule can be repeated in a way that the two tasks are scheduled on the faster processor in turn. As shown in Fig. 5, the maximum tardiness of τ_1 is 3 time units ($\tau_{1,3}$, $\tau_{1,7}$, ...); the maximum tardiness of τ_2 is 2 time units ($\tau_{2,1}$, $\tau_{2,5}$, $\tau_{2,9}$,...).

This suggests that, to achieve SRT-optimality on uniform heterogeneous multiprocessors by a non-preemptive scheduler, a non-work-conserving mechanism is needed.

Furthermore, for some other instantiation (or release pattern) of this task set τ, a different non-preemptive schedule may be needed. Thus, another interesting question arises that whether clairvoyance is necessary to obtain optimal non-preemptive schedulers. We defer further study on this to future work.

6 Conclusion

Motivated by the fact that both the preemptive and non-preemptive GEDF schedulers are known to be SRT-optimal on identical multiprocessors, we studied these optimality results while extending the underlying platform from being identical to being uniform. We proved that the SRT-optimality of preemptive GEDF is preserved on two-processor uniform platforms, but the question of SRT-optimality of preemptive GEDF remains open for larger platforms. On the other hand, we showed that no work-conserving non-preemptive scheduler, including non-preemptive GEDF, can be SRT-optimal on uniform platforms, even if the number of processors is limited to two.

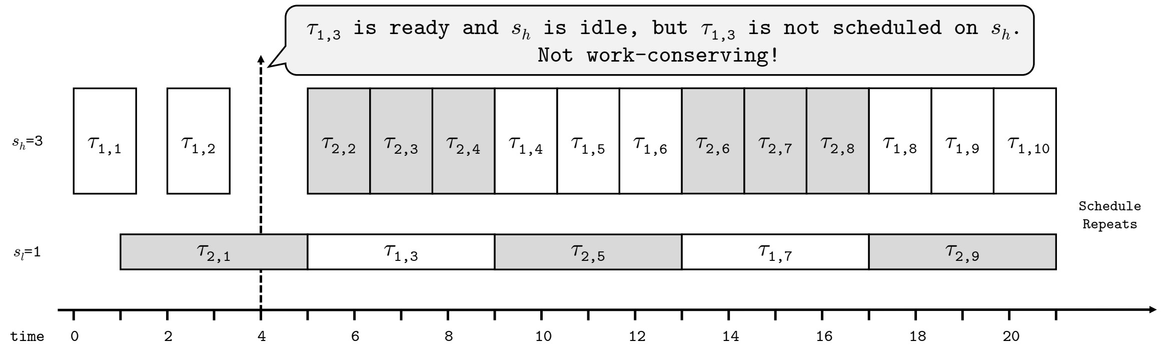


Figure 5: A non-preemptive schedule for the system in Sec. 5. Note that the deadline tardiness is upper bounded by 3 time units.

Future work. Directly from the discussion in Sec. 4 and the negative results in Sec. 5, there are two open problems we would like to address. First, we would like to prove or disprove the SRT-optimality of preemptive GEDF on more than three uniform processors. Perhaps new proof techniques and frameworks are needed to handle the situation of being scheduled on "not fast enough" processors. Second, we would like to develop non-work-conserving schemes and design a non-preemptive scheduling algorithm that is SRT-optimal on uniform platforms.

References

[1] G. Buttazzo. Real-time systems: Achievements and perspectives. Award Speech in *35th RTSS*, 2014. Slides: http://2014.rtss.org/wp-content/uploads/2014/12/Buttazzo-award-talk-RTSS14.pdf.

[2] U. Devi and J. Anderson. Tardiness bounds for global EDF scheduling on a multiprocessor. In *26th RTSS*, 2005.

[3] S. Dhall and C. Liu. On a real-time scheduling problem. *Operations research*, 26(1):127–140, 1978.

[4] J. Erickson and J. Anderson. Fair lateness scheduling: Reducing maximum lateness in G-EDF-like scheduling. In *24th ECRTS*, 2012.

[5] J. Erickson, U. Devi, and S. Baruah. Improved tardiness bounds for global EDF. In *22nd ECRTS*, 2010.

[6] S. Funk. *EDF Scheduling on Heterogeneous Multiprocessors*. PhD thesis, University of North Carolina, Chapel Hill, NC, 2004.

[7] S. Funk, J. Goossens, and S. Baruah. On-line scheduling on uniform multiprocessors. In *22nd RTSS*, 2001.

[8] H. Leontyev and J. Anderson. Generalized tardiness bounds for global multiprocessor scheduling. In *28th RTSS*, 2007.

[9] H. Leontyev and J. Anderson. Tardiness bounds for EDF scheduling on multi-speed multicore platforms. In *13th RTCSA*, 2007.

[10] M. Pinedo. *Scheduling, Theory, Algorithms, and Systems*. Prentice Hall, 1995.

[11] G. Tong and C. Liu. Supporting soft real-time sporadic task systems on heterogeneous multiprocessors with no uilitzation loss. *IEEE Transactions on Parallel and Distributed Systems*, to appear, 2015.

[12] K. Yang and J. Anderson. Optimal GEDF-based schedulers that allow intra-task parallelism on heterogeneous multiprocessors. In *12th ESTIMedia*, 2014.

[13] K. Yang and J. Anderson. Soft real-time semi-partitioned scheduling with restricted migrations on uniform heterogeneous multiprocessors. In *22nd RTNS*, 2014.

[14] K. Yang and J. Anderson. An optimal semi-partitioned scheduler for uniform heterogeneous multiprocessors. In *27th ECRTS*, 2015.