# Optimal Online Multiprocessor Scheduling of Sporadic Real-Time Tasks is Impossible

Nathan Fisher
Wayne State University

Joël Goossens
Université Libre de Bruxelles

Sanjoy Baruah
The University of North Carolina at Chapel Hill

## Abstract

*Optimal online scheduling algorithms are known for sporadic task systems scheduled upon a single processor. Additionally, optimal online scheduling algorithms are also known for restricted subclasses of sporadic task systems upon an identical multiprocessor platform. The research reported in this article addresses the question of existence of optimal online multiprocessor scheduling algorithms for general sporadic task systems. Our main result is a proof of the impossibility of optimal online scheduling for sporadic task systems upon a system comprised of two or more processors. The result is shown by finding a sporadic task system that is feasible on a multiprocessor platform that cannot be correctly scheduled by any possible online, deterministic scheduling algorithm. Since the sporadic task model is a subclass of many more general real-time task models, the nonexistence of optimal scheduling algorithms for the sporadic task systems implies nonexistence for any model which generalizes the sporadic task model.*

**Keywords:** *Real-time scheduling; Multiprocessor systems; Sporadic task model; Optimal scheduling algorithms.*

## 1 Introduction

The *sporadic task model* [18, 16] has received tremendous research attention over the years for its usefulness in modeling recurring processes for hard-real-time systems. A *sporadic task* $\tau_i = (e_i, d_i, p_i)$ is characterized by a *worst-case execution requirement* $e_i$, a *(relative) deadline* $d_i$, and a *minimum inter-arrival separation* $p_i$, which is, for historical reasons, also referred to as the *period* of the task. Such a sporadic task generates a potentially infinite sequence of jobs, with successive job-arrivals separated by at least $p_i$ time units. Each job has a worst-case execution requirement equal to $e_i$ and a deadline that occurs $d_i$ time units after its arrival time. A *sporadic task system* $\tau$ is a collection of such sporadic tasks.

Two significant factors contribute to the popularity of the sporadic task model in real-time system design. One factor is the generality of the sporadic task model. The sporadic task model is an extension of an earlier task model known as the *Liu and Layland (LL) task model* [17]. An LL task, $\tau_i$, is only specified by an worst-case execution requirement $e_i$ and a period $p_i$. The relative deadline is implicit in the period parameter (i.e., a job of an LL task has absolute deadline $p_i$ time units after its arrival). The sporadic task model is, thus, a generalization of the LL task model, and, in fact, LL tasks are a subclass of sporadic task systems sometimes referred to as *implicit-deadline* sporadic task systems. Other subclasses of sporadic task systems include *constrained-deadline* sporadic task systems where each task has $d_i \leq p_i$ and *arbitrary-deadline* sporadic task systems where no constraint is imposed upon the relationship between a task's deadline and period.

The development of effective and efficient scheduling algorithms and associated analytical techniques for single processor systems is another factor in the sporadic task model's popularity. For instance, the earliest-deadline-first (EDF) scheduling algorithm is known to be *optimal* for arbitrary collections of independent jobs scheduled upon uniprocessor platforms [8]. This optimality result holds for both sporadic task systems and LL task systems on uniprocessors. The notion of optimality for real-time systems is explained in the following: a task system $\tau$ is said to be *feasible* on a processing platform, if, for any legal job arrival sequence of $\tau$, there exists a schedule for $\tau$ on the processing platform in which each job successfully completes execution by its deadline. For any task system $\tau$ that is feasible on a given processing platform, an optimal scheduling algorithm is guaranteed to generate a schedule for $\tau$ which meets all deadlines. In addition to the existence of optimal scheduling algorithms for sporadic task systems, exact, pseudo-polynomial-time techniques are known for determining whether a given sporadic task system is feasible upon a preemptive single processor platform [7]. Such techniques are known as *feasibility analysis*. A related analysis technique, known as *schedulability analysis*, determines whether a given scheduling algorithm will correctly schedule a task system to meet all deadlines on a processing platform. Relatively efficient, exact schedulability tests have been developed for various scheduling algorithms on uniprocessor platforms.

The success of the sporadic task model for real-time system design on single processor systems has motivated research on scheduling algorithms and feasibility/schedulability analysis for sporadic task systems upon multiprocessor platforms. Unfortunately, most results from uniprocessor scheduling of sporadic task systems do not trivially extend to the multiprocessor setting. For instance, it is known that EDF is a suboptimal scheduling algorithm for even LL tasks on multiprocessor platforms [10]. However, optimal scheduling approaches for LL task systems have been developed [13, 6, 20].

Since LL tasks are a subclass of sporadic task systems, the non-optimality result for EDF [10] extends trivially to sporadic task systems on multiprocessor platforms. The question that this article addresses is: *does there exist an algorithm which is guaranteed to successfully schedule any feasible sporadic task system on a multiprocessor platform*? In other words, does there exist optimal scheduling algorithms for sporadic task model? For LL task systems, the answer to that question is "yes," due to the existence of optimal scheduling approaches (referred to in the preceding paragraph). For arbitrary collections of independent jobs where job arrival-times are not known *a priori*, Hong and Leung [12] and Dertouzos and Mok [9], independently, showed that the answer is "no"; i.e., optimal online scheduling of arbitrary collections of independent jobs is impossible. In terms of generality, the sporadic task model lies between the LL task model (any LL task system is also a sporadic task system) and the

arbitrary collections of independent jobs setting (any collection of jobs generated by a sporadic task system is also a legal collection of independent jobs). As we will illustrate later in this article, the multiprocessor optimality result for LL task systems and the non-optimality result do not directly apply to the sporadic task systems. Thus, the above question cannot be answered by application of prior results.

The main contribution of this article answers the above open question in the negative: *optimal online multiprocessor scheduling of sporadic task systems is impossible*. We, in fact, show a slightly stronger result that optimal online multiprocessor scheduling of constrained-deadline sporadic task systems is impossible. The impossibility result for constrained-deadline sporadic task systems immediately implies that optimal online scheduling of any task model that generalizes the constrained-deadline sporadic task model is impossible, as well. Therefore, even a slight amount of generalization from the LL task model (the sporadic task model simply adds a relative deadline parameter to the task specification) causes the existence of optimal scheduling algorithms to disappear.

This article is organized as follows. Section 2 presents the formal models and notation that we use for describing real-time work, task systems, processing platforms, and scheduling algorithms, Section 3 illustrates (via examples) the inapplicability of prior multiprocessor optimality results to the multiprocessor scheduling of sporadic task systems. Section 4 proves that optimal online multiprocessor scheduling of sporadic and more general task systems is impossible. The proof given in Section 4 relies upon an example task system that is assumed to be feasible upon a multiprocessor platform; Section 5 proves that this example task system is, in fact, feasible.

## 2 Model and Notation

### 2.1 Real-Time Instances

Throughout this article, we will characterize a real-time *job* $J_i$ by a three-tuple $(A_i, E_i, D_i)$: an *arrival time* $A_i$, an execution requirement $E_i$, and a relative deadline $D_i$. The interpretation of these parameters is that $J_i$ arrives $A_i$ time units after system start-time (assumed to be zero) and must execute for $E_i$ time units over the time interval $[A_i, A_i + D_i)$. $A_i$ is assumed to be a non-negative real number while both $E_i$ and $D_i$ are positive real numbers. The interval $[A_i, A_i + D_i)$ is referred to as $J_i$'s *scheduling window*. A job $J_i$ is said to be *current* at time $t$ if $t \in [A_i, A_i + D_i)$. A current job is *active* at time $t$, if it has not completed execution by time $t$.

We denote a real-time *instance* $I$ as a finite or infinite collection of jobs $I = \{J_1, J_2, \ldots\}$. $\mathcal{F}(I)$ denotes a real-time instance *family* with representative real-time instance $I$. For each job $J_i'$ in real-time instance $I' \in \mathcal{F}(I)$, there is a job $J_i$ in instance $I$ with the same release time and deadline; however, the execution of $J_i'$ cannot exceed the execution time of $J_i$. More formally, $I' \in \mathcal{F}(I)$ if and only if

$$\forall J_i' \in I', \exists J_i \in I :: (A_i' = A_i) \wedge (D_i' = D_i) \wedge (E_i' \leq E_i).$$

Informally, $\mathcal{F}(I)$ represents a set of related real-time instances with $I$ being the most "temporally constrained" of the set.

**Example 1** Consider a real-time instance $I = \{(0, 2, 3), (5, 4, 5), (6, 2, 4)\}$. $\mathcal{F}(I)$ includes any instance $I' = \{(0, x, 3), (5, y, 5), (6, z, 4)\}$ such that $0 \leq x \leq 2$, $0 \leq y \leq 4$, and $0 \leq z \leq 2$. ■

### 2.2 Real-Time Task Models

In some simpler real-time systems, it may be possible to completely specify the real-time instance $I$ prior to system run-time (i.e., the system designer has complete knowledge of each $J_i \in I$). However, in systems with a large (or infinite) number of real-time jobs or systems that exhibit dynamic behavior, explicitly specifying each job, prior to system run-time, may be impossible or unreasonable. Fortunately, for systems where jobs may repeatedly

occur there is a more succinct representation of the repeating jobs via specification in some *recurrent task model*. A *task model* is the format and rules for specifying a task system. We may represent a set of repeating or related jobs by a recurrent task $\tau_i$ specified according to the model $M$ (e.g., the sporadic task model). For every execution of the system, $\tau_i$ will generate a (possibly infinite) collection of real-time jobs.

Several recurrent tasks can be composed together into a recurrent task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$. The letter $n$ will denote the number of tasks in a task system. Every system execution of task system $\tau$ will result in the generation of a real-time instance $I$. We will denote the set of real-time instances that $\tau$ can legally generate as $\mathcal{I}^{\mathrm{M}}(\tau)$. Based on the real-time instances that $\tau$ generates, we can classify $\tau$ as either *completely specified* or *partially-specified*. If the arrival-time and deadline parameters of each job $J_i \in I$ can be determined prior to system run-time, $\tau$ is a completely-specified task system. However, for many real-time systems, it is not possible to know beforehand what real-time instance will be generated by the system during run-time. Furthermore, completely-specified systems are incapable of handling changes in real-time workloads. To overcome the fragile and inflexible nature of completely-specified task systems, a designer may instead consider partially-specified tasks systems.[1] The focus of this article is on partially-specified task systems.

Partially-specified task systems permit that different executions of the same system may result in different real-time instances (with different job arrival times) being generated. The specification for a partially-specified task system includes a set of constraints that any generated real-time instance must satisfy; in general, such a system may legally generate infinitely many different real-time instances, each of which satisfies the constraints placed upon their generation. Each such real-time instance may also have infinitely many jobs.

Let $M$ and $M'$ be task models. We say that task model $M'$ *generalizes* task model $M$, if for every task system $\tau$ specified in model $M$ there exists a task system $\tau'$ specified in model $M'$ such that

$$I \in \mathcal{I}^{\mathrm{M}}(\tau) \Leftrightarrow I \in \mathcal{I}^{\mathrm{M}'}(\tau').$$

That is, for all task systems $\tau$ that can be specified in task model $M$, there is a task system $\tau'$ specified in task model $M'$ that can generate exactly the same real-time instances as $\tau$. In the remainder of this subsection, we describe the Liu and Layland task model and sporadic task model in this more formal context.

§ **Liu and Layland (LL) Task Model (Implicit-Deadline Sporadic Task Model).** As mentioned in the introduction, the behavior of a LL task $\tau_i$ can be characterized by a two-tuple $(e_i, p_i)$. As with the periodic task model, $e_i$ indicates the worst-case execution time of any job generated by task $\tau_i$. The $p_i$ parameter indicates the *minimum inter-arrival time* between successive jobs of $\tau_i$ (note $p_i$ denoted the *exact* inter-arrival time for periodic tasks). Let $\mathcal{J}^{\mathrm{LL}}_{\mathrm{WCET}}(\tau_i)$ be a collection of real-time instances such that jobs of each real-time instance are generated by LL task $\tau_i$ satisfying the minimum inter-arrival constraint and requiring the worst-case possible execution time; i.e., $I_{\tau_i}$ is a member of $\mathcal{J}^{\mathrm{LL}}_{\mathrm{WCET}}(\tau_i)$ if and only if for all $J_k \in I_{\tau_i}$ the following constraints are satisfied:

$$(E_k = e_i) \wedge (D_k = p_i) \wedge ((\exists J_{k+1} \in I_{\tau_i} \setminus \{J_k\} : A_{k+1} \geq A_k) \Rightarrow (A_{k+1} - A_k \geq p_i)). \tag{1}$$

The set of real-time instances that a LL task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ can generate (with worst-case possible execution time) is equal to

$$\mathcal{I}^{\mathrm{LL}}_{\mathrm{WCET}}(\tau) \stackrel{\text{def}}{=} \left\{ \bigcup_{i=1}^{n} I_{\tau_i} \;\middle|\; (I_{\tau_1}, I_{\tau_2}, \ldots, I_{\tau_n}) \in \prod_{i=1}^{n} \mathcal{J}^{\mathrm{LL}}_{\mathrm{WCET}}(\tau_i) \right\}. \tag{2}$$

Thus, the set of real-time instances generated by LL task system $\tau$ is

$$\mathcal{I}^{\mathrm{LL}}(\tau) = \bigcup_{I_j \in \mathcal{I}^{\mathrm{LL}}_{\mathrm{WCET}}(\tau)} \mathcal{F}(I_j). \tag{3}$$

[1] A partially-specified task system is sometimes referred to as *non-concrete* [14].

**Example 2** Consider the following LL task system: $\tau = \{\tau_1 = (2,4), \tau_2 = (3,10)\}$. Examples of sets of jobs in $\mathcal{J}^{\mathrm{LL}}_{\mathrm{WCET}}(\tau_1)$ are $\{(0,2,4),(4,2,4),(8,2,4),\ldots\}$, $\{(0,2,4),(5,2,4),(9,2,4)\}$, and $\{(0,2,4),(6,2,4),(10,2,4),\ldots\}$; examples of sets of jobs in $\mathcal{J}^{\mathrm{LL}}_{\mathrm{WCET}}(\tau_2)$ are $\{(0,3,10),(10,3,10),(20,3,10),\ldots\}$, $\{(1,3,10),(15,3,10),(25,3,10),\ldots\}$, and $\{(5,3,10),(15,3,10),(25,3,10),\ldots\}$. ■

§ **Sporadic Task Model.** The LL task model allows for flexibility in the job arrival times for a task $\tau_i$; however, the model is still somewhat restrictive in forcing the deadline of each job generated by $\tau_i$ to be equal to the minimum inter-arrival parameter $p_i$. It is easy to imagine scenarios where the deadline of a job is not correlated with the minimum inter-arrival: for example, in a car's brake system the minimum time between braking events may be considerably larger than the required braking-reaction time (i.e., deadline for halting the car). The sporadic task model generalizes the LL task model by adding a *relative deadline* parameter $d_i$ to the specification for a task. Recall that a sporadic task $\tau_i$ is specified by the three-tuple $(e_i, d_i, p_i)$. Let $\mathcal{J}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_i)$ be a collection of real-time instances that are jobs generated by sporadic task $\tau_i$ satisfying the minimum inter-arrival constraint and requiring the worst-case possible execution time; i.e., $I_{\tau_i}$ is a member of $\mathcal{J}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_i)$ if and only if for all $J_k \in I_{\tau_i}$

the following constraints are satisfied:

$$(E_k = e_i) \wedge (D_k = d_i) \wedge ((\exists J_{k+1} \in I_{\tau_i} \setminus \{J_k\} : A_{k+1} \geq A_k) \Rightarrow (A_{k+1} - A_k \geq p_i)) . \tag{4}$$

(Note that the only difference from Equation 1 for LL jobs is that the $D_k$ parameter for each job $J_k$ is set to $d_i$). The set of real-time instances that a sporadic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ can generate (with worst-case possible execution times) is

$$\mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau) \stackrel{\text{def}}{=} \left\{ \bigcup_{i=1}^{n} I_{\tau_i} \;\middle|\; (I_{\tau_1}, I_{\tau_2}, \ldots, I_{\tau_n}) \in \prod_{i=1}^{n} \mathcal{J}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_i) \right\}. \tag{5}$$

Thus, the set of real-time instances generated by sporadic task system $\tau$ is

$$\mathcal{I}^{\mathrm{S}}(\tau) = \bigcup_{I_j \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau)} \mathcal{F}(I_j). \tag{6}$$

Observe that for any LL task system $\tau = \{\tau_1 = (e_1, p_1), \ldots, \tau_n = (e_n, p_n)\}$ we can represent the same task system in the sporadic model by the sporadic task system $\tau' = \{\tau'_1 = (e_1, p_1, p_1), \ldots, \tau_n = (e_n, p_n, p_n)\}$. It is easy to see that $\mathcal{I}^{\mathrm{LL}}(\tau) = \mathcal{I}^{\mathrm{S}}(\tau')$; therefore, the sporadic task model generalizes the LL task model.

§ **More General Task Models.** There are other known real-time task models more general than the sporadic task model. For example, the *generalized multiframe (GMF) task model* [5] allows for a task to generate sequence of jobs with heterogenous separation, relative deadlines, and worst-case execution parameters. Another general task model, known as the *recurring real-time task model* [4], allows for conditional generation of job sequences for a task. Both of these models generalize the sporadic task model. Thus, the impossibility of optimal online multiprocessor scheduling algorithms for sporadic task systems implies the impossibility of optimal scheduling algorithms for these more general task models, as well.

### 2.3 Machine Model

This article focuses on the real-time scheduling upon multiprocessor platforms. More specifically, we will be concentrating on scheduling upon a class of multiprocessor platforms known as the *identical multiprocessors*. The identical multiprocessor model assumes that each processor in the platform has identical processing capabilities and speed. We denote the multiprocessor platform by $\Pi$ and assume $\Pi$ is comprised of $m$ identical processors $\pi_1$, $\pi_2$, …, $\pi_m \in \Pi$. Recall from the beginning of this paper that each job corresponds to the execution of a sequential

segment of code by the processing platform. For each model introduced in the previous subsection, a real-time task has associated worst-case execution requirement parameter(s). These execution requirements represent the worst-case cumulative amount of execution time that a job generated by the task requires to execute to completion on the processing platform.

§ **Some Assumptions.** We will assume that each processor has unit-speed. We will assume that jobs are preemptable at arbitrary times with no additional cost. Furthermore, we allow scheduling algorithms which *migrate* jobs between processor; that is, a job may execute on different processors over its scheduling window; however, job-level parallelism is not permitted (i.e., a job may not execute concurrently with itself on two or more processors simultaneously). We will make the simplifying assumption that migration does not incur any additional penalty or execution. Throughout this article, we will also assume that tasks are *independent* of each other; that is, the execution of a job of one task is not contingent upon the status of a job of another task (e.g., blocking on shared resources is not permitted). Most of the above assumptions are not limiting; in fact, the nonexistence of optimal online multiprocessor scheduling algorithms for sporadic task systems under this simplified setting implies the non-existence of optimal scheduling algorithms when the assumptions on preemption, migration, and task independence are removed.

### 2.4 Real-Time Scheduling Algorithms

When executing a real-time application, the real-time scheduling algorithm must determine which current jobs are executing on the processing platform at every time instant. At an abstract level, the real-time scheduling algorithm determines the interleaving of execution for jobs of any real-time instance $I$ on the processing platform $\Pi$. The interleaving of execution of $I$ on $\Pi$ is known as a *schedule*. The goal of a real-time scheduling algorithm is to produce a schedule that ensures that every job of $I$ is allocated the processor (i.e., executes) for its execution requirement during its scheduling window. In this subsection, we give some formal definitions for real-time scheduling algorithm concepts.

We can formally define the schedule $S$ for real-time instance $I$ as a function of the processor and time.

**Definition 1 (Schedule Function)** *Let $S_I(\pi_k, t)$ be the job of $I$ scheduled at time $t$ on processor $\pi_k \in \Pi$; $S_I(\pi_k, t)$ is $\perp$ if there is no task scheduled at time $t$ (i.e., $S_I : \Pi \times \mathbb{R}^+ \mapsto I \cup \{\perp\}$). Let $\mathbb{S}_{I,\Pi}$ be the set of all possible schedule functions over real-time instance $I$ and platform $\Pi$.*

It is sometimes useful to view the behavior of a single job of a real-time instance $I$ in schedule $S_I$. The following definition allows us to characterize the schedule $S_I$ with respect to task $J_i$.

**Definition 2 (Job-Schedule Function)** *$S_I(\pi_k, t, J_i)$ is an indicator function denoting whether $J_i$ is scheduled at time $t$ on processor $\pi_k$ for schedule $S_I$. In other words,*

$$S_I(\pi_k, t, J_i) \stackrel{def}{=} \begin{cases} 1\,, & \text{if } S_I(\pi_k, t) = J_i \\ 0\,, & \text{otherwise.} \end{cases} \tag{7}$$

A scheduling algorithm makes decisions about the order in which jobs of a real-time instance should execute. For systems that are partially-specified, an *online* algorithm is appropriate to handle dynamic job arrivals. For any time $t$, an online real-time scheduling algorithm decides the set of jobs that will be executed on $\Pi$ at time $t$ based on prior decisions and the status of jobs released at or prior to $t$. An online scheduling algorithm does not have specific information on the release of jobs after time $t$ (i.e., future jobs arrival times are unknown). This article focuses on deterministic online, real-time multiprocessor scheduling algorithms.

At an abstract level, a real-time scheduling algorithm[2] $\mathcal{A}$ (either static or offline) on platform $\Pi$ is a higher-order function[3] from real-time instances to schedules over $\Pi$ — i.e., $\mathcal{A} : \mathcal{I}^{\mathrm{M}}(\tau) \rightarrow \bigcup_{I \in \mathcal{I}} \mathbb{S}_{I,\Pi}$. Let $I_{\leq t} \stackrel{\text{def}}{=} \{J_i \in I | A_i \leq t\}$; that is, $I_{\leq t}$ is the set of jobs of $I$ that arrive prior to or at time $t$. For an online scheduling algorithm $\mathcal{A}$, $I_{\leq t}$ represents the set of jobs that $\mathcal{A}$ has knowledge of at time $t$ (i.e., $\mathcal{A}$ knows the arrival time, execution requirement, and deadline parameters of the jobs of $I_{\leq t}$, but not other jobs of $I$). Up until time $t$, algorithm $\mathcal{A}$ has made scheduling decisions without specific knowledge of jobs arriving after time $t$; furthermore, jobs arriving after $t$ cannot have an effect on the schedule generated by $\mathcal{A}$ from time zero to $t$. In other words, for an online scheduling algorithm future jobs cannot change past scheduling decisions.

**Definition 3 (Deterministic Online Scheduling Algorithm)** *For any $I \in \mathcal{I}^{\mathrm{M}}(\tau)$, let $S_I^{\mathcal{A}}$ be the schedule produced by algorithm $\mathcal{A}$ for real-time instance $I$ and platform $\Pi$. An online real-time scheduling algorithm must satisfy the following constraint: for all $I, I' \in \mathcal{I}^{\mathrm{M}}(\tau)$ and for all $t > 0$,*

$$(I_{\leq t} = I'_{\leq t}) \Rightarrow \left(\forall t'(0 \leq t' \leq t), \forall \pi_k \in \Pi :: S_I^{\mathcal{A}}(\pi_k, t') = S_{I'}^{\mathcal{A}}(\pi_k, t')\right). \tag{8}$$

Beyond restricting our attention to deterministic, online scheduling algorithms and algorithms that forbid job-level parallelism, we do not make any other restrictions on the scheduling algorithm.

### 2.5 Feasible Real-Time Task Systems

The definition of "optimal scheduling algorithm" makes use of the notion of a task system being feasible upon a processing platform: an optimal scheduling algorithm can correctly schedule any feasible task system. Thus, we need to formalize what we mean by "feasible task system." This subsection defines "feasible" and other related concepts.

When evaluating a real-time system, it is sometimes useful to describe the amount of "work" (execution) that a job does over a specified interval in a given schedule. The next definition defines the amount of "processor time" that a job receives over a given interval.

**Definition 4 (Work Function)** *$W(S_I, \pi_k, J_i, t_1, t_2)$ denotes the amount of processor time on $\pi_k$ that $J_i$ receives from schedule $S_I$ over the interval $[t_1, t_2)$. In other words,*[4]

$$W(S_I, \pi_k, J_i, t_1, t_2) \stackrel{def}{=} \int_{t_1}^{t_2} S_I(\pi_k, t, J_i) dt. \tag{9}$$

We can use a *system-work function* to describe the cumulative work done by all jobs of a real-time instance over a specified time interval in a given schedule.

**Definition 5 (System-Work Function)** *$W_I(S_I, t_1, t_2)$ denotes the amount of processor time (over all processors of $\Pi$) received by all jobs of $I$ in schedule $S_I$ over the interval $[t_1, t_2)$.*

$$W_I(S_I, t_1, t_2) \stackrel{def}{=} \sum_{\pi_k \in \Pi} \sum_{J_i \in I} W(S_I, \pi_k, J_i, t_1, t_2). \tag{10}$$

[2] We will slightly abuse notation and use $\mathcal{A}$ to refer to both the scheduling algorithm and the function.

[3] A *higher-order function* has a function space as either the domain or range.

[4] Since $S_I(\pi_k, t, J_i)$ is potentially discontinuous at an infinite number of points, $\int_{t_1}^{t_2} S_I(\pi_k, t, J_i) dt$ denotes a Lebesgue integral [15] and not a Riemann integral.

Not all functions from $\Pi \times \mathbb{R}^+$ to $I$, for a given real-time instance $I$, represent valid executions of a real-time system that could generate the instance $I$. In particular, we must ensure the following: a job can only execute during its scheduling window, a job cannot execute concurrently with itself on two or more processors, and a job must execute for $E_i$ time units in its scheduling window to meet its deadline. Using Definitions 1 through 5, we can now formally define a *valid* schedule $S_I$ with respect to a real-time instance $I$:

**Definition 6 (Valid Schedule)** *$S_I \in \mathbb{S}_{I,\Pi}$ is valid (with respect to jobs of some real-time instance $I$ and platform $\Pi$) if and only if the following three conditions are satisfied:*

1. *For any $J_i \in I$, if $t < A_i$ or $t > A_i + D_i$ then $S_I(\pi_k, t) \neq J_i$ for all $\pi_k \in \Pi$ (i.e., a job cannot execute while it is outside its scheduling window). For this article, will assume that two different jobs of the same task may execute concurrently on different processors (i.e., intra-task parallelism is allowed, but intra-job parallelism is forbidden).*
2. *If $S_I(\pi_i, t) \neq \perp$ and $S_I(\pi_j, t) \neq \perp$ then $S_I(\pi_i, t) \neq S_I(\pi_j, t)$ for all $t \in \mathbb{R}^+$ and $\pi_i \neq \pi_j \in \Pi$ (i.e., a job may not execute concurrently with itself).*
3. *For all $J_i \in I$, $W_I(S_I, J_i, A_i, A_i + D_i) = E_i$ (i.e., each job receives processing time on $\Pi$ equal to its execution requirement between its release time and deadline).*

Recall that a recurrent task system can potentially generate infinitely different distinct real-time instances over different executions of the system. Informally, a recurrent task system $\tau$ is *feasible* on processing platform $\Pi$ if and only if for every possible real-time instance there exists a way to meet all deadlines. If there is a way for a real-time instance $I$ to meet all deadlines, we say that $I$ is a *feasible instance* on processing platform $\Pi$.

**Definition 7 (Feasible Instance)** *A real-time instance $I$ is* feasible *on platform $\Pi$ if and only if there exists $S_I \in \mathbb{S}_{I,\Pi}$ such that $S_I$ is valid.*

We may extend the definition of feasible real-time instances to recurrent task systems.

**Definition 8 (Feasible Task System)** *Recurrent task system $\tau$ in task model $\mathrm{M}$ is* feasible *on platform $\Pi$ if and only if for all $I \in \mathcal{I}^{\mathrm{M}}(\tau)$, $I$ is a feasible instance on $\Pi$.*

## 3 Inapplicability of Prior Optimality Results for Multiprocessor Real-Time Scheduling

The nonexistence of optimal online multiprocessor real-time scheduling algorithms for arbitrary collection of jobs has been known since the late 1980s [12, 9]. However, as mentioned in the introduction, these results do not imply the nonexistence of optimal multiprocessor scheduling algorithms for sporadic task systems. In this section, we will briefly review the Dertouzos and Mok [9] proof of impossibility for optimal scheduling of arbitrary collection of real-time jobs and discuss why this result does not apply to sporadic task systems. We will omit a discussion of the Hong and Leung result [12], since a nearly identical argument will show that their results also do not apply to the sporadic task model setting. The following is a restatement of the main result from [9].

**Theorem 1 (from Dertouzos and Mok [9])** *For two or more processors, no online scheduling algorithm can be optimal for arbitrary collections of real-time jobs without complete* a priori *knowledge of the absolute deadlines, execution time, and arrival time of each job.*

Why does the above theorem not imply that sporadic task systems have no optimal multiprocessor scheduling algorithm? Intuitively, the reason is that for arbitrary real-time instances an optimal scheduling algorithm must be

able to correctly schedule any feasible real-time instances. While for sporadic task systems, an optimal scheduling algorithm must correctly schedule only feasible real-time instances that may be legally generated by a sporadic task system. To more clearly illustrate this point let us consider the following lemma from [9] used to prove Theorem 1.

**Lemma 1 (from Dertouzos and Mok [9])** *For two or more processors, no online scheduling algorithm for arbitrary collections of real-time jobs without complete* a priori *knowledge of the arrival time of each job.*

The above lemma is proven in [9] by finding a set of feasible real-time instances that are identical up until a some time $t$ that would cause any deterministic online scheduling algorithm to miss a deadline after time $t$. Below is the example set of feasible real-time instances used by Dertouzos and Mok [9] to prove Lemma 1.

**Example 3** Define the following set of real-time instances.

$$\begin{aligned} I_1 &\stackrel{\text{def}}{=} \{J_1 = (0, 2, 4), J_2 = (0, 1, 1), J_3 = (0, 1, 2)\}, \\ I_2 &\stackrel{\text{def}}{=} \{J_4 = (1, 1, 1), J_5 = (1, 1, 1)\}, \\ I_3 &\stackrel{\text{def}}{=} \{J_6 = (1, 2, 2), J_7 = (1, 2, 2)\}, \\ I_A &\stackrel{\text{def}}{=} I_1 \cup I_2, \\ I_B &\stackrel{\text{def}}{=} I_1 \cup I_3. \end{aligned} \tag{11}$$

Consider how any online, deterministic scheduling algorithm $\mathcal{A}$ would execute real-time instances $I_A$ or $I_B$ on platform $\Pi = \{\pi_1, \pi_2\}$ comprised of two identical unit-speed processors. To simplify the presentation of the example, let us assume that $\mathcal{A}$ only makes scheduling decisions at integer time instants (i.e., preemptions will not occur at non-integer time instants); the lemma holds even when we remove this simplifying assumption. If $\mathcal{A}$ does not know the arrival times of each job prior to their arrival, at time zero algorithm $\mathcal{A}$ can only make a scheduling decision based upon the knowledge of the set of jobs in $I_1$ (for scheduling either $I_A$ or $I_B$). Real-time instances $I_A$ and $I_B$ appear to be identical to $\mathcal{A}$ for all times in the interval $[0, 1)$. However, $\mathcal{A}$ must make a decision about what set of jobs will execute over $[0, 1)$ on the two processors of $\Pi$ without knowledge of the jobs that may arrive at time-instant one (i.e., at time zero, $\mathcal{A}$ does not know whether it is executing $I_A$ or $I_B$). Obviously, $\mathcal{A}$ must execute job $J_2$ on some processor (w.l.o.g., assume $\pi_1$) over the interval $[0, 1)$ for $J_2$ to meet its deadline at time-instant one. The non-obvious choice is what should execute on $\pi_2$ over $[0, 1)$? There are three possible choices:

1. $\mathcal{A}$ executes $J_1$ on $\pi_2$ over $[0, 1)$.
2. $\mathcal{A}$ executes $J_3$ on $\pi_2$ over $[0, 1)$.
3. $\mathcal{A}$ executes no job on $\pi_2$ over $[0, 1)$.

If $\mathcal{A}$ executes $J_1$ over $[0, 1)$, real-time instance $I_A$ would miss a deadline at time-instant two; observe in this scenario $J_3$, $J_4$, and $J_5$ must execute exactly continuously over $[1, 2)$ to meet their deadline, but there are only two available processors. For a similar reason, $I_A$ would also miss a deadline at time-instant two, if $\mathcal{A}$ chose not to execute a job on $\pi_2$ over $[0, 1)$. If $\mathcal{A}$ instead executes $J_3$ over $[0, 1)$, real-time instance $I_B$ would miss a deadline at time-instant three, since $J_1$, $J_6$, and $J_7$ require continuous execution over $[1, 3)$. The reader should observe that $I_A$ and $I_B$ are both feasible on two processors (i.e., a valid schedule may be found for both instances). However, the above case analysis shows that for any choice made by $\mathcal{A}$ at time zero (without knowledge of future job arrivals), there exist a feasible set of future job arrivals that will cause $\mathcal{A}$ to miss a deadline. Thus, optimal online scheduling is impossible for arbitrary collections of real-time jobs on two processors. This example may easily be extended to an arbitrary number of processors. ■

For the above example to imply the non-existence of optimal online multiprocessor scheduling algorithms for sporadic task systems, we must show that $I_A$ and $I_B$ correspond to legal real-time instances generated by a sporadic task system $\tau$ that is feasible on two processors. One possible sporadic task system that could generate both the real-time instances $I_A$ and $I_B$ is

$$\tau \stackrel{\text{def}}{=} \{\tau_1 = (2, 4, \infty), \tau_2 = \tau_3 = \tau_4 = (1, 1, \infty), \tau_5 = (1, 2, \infty), \tau_6 = \tau_7 = (2, 2, \infty)\}. \tag{12}$$

The above task system allows each job of $I_A \cup I_B$ to be generated by a different task. Real-time instances $I_A$ and $I_B$ satisfy the constraints of Equation 4 for task system $\tau$. However, $\tau$ is not feasible on two processors since the real-time instance where each task of $\tau$ generates a job at time-instant zero is also a legal real-time instance; such an instance requires that at least five jobs execute continuously over $[0, 1)$! Other possible groupings of jobs to task also appear to result in a sporadic task system that is infeasible on two processors. The difficulty in finding a feasible task system that can generate both $I_A$ and $I_B$, suggests that such a sporadic task system may not exist. Thus, Lemma 1 and Theorem 1 do not directly imply anything about the existence of an optimal online algorithm for sporadic task systems. A similar argument may be used to argue about the inapplicability of the results of Hong and Leung [12] to sporadic task systems. We should also point out that the main result of Section 4 (Theorem 3) implies the impossibility of optimal scheduling for arbitrary collections of real-time jobs without knowledge of future arrival times. Thus, our results can be considered a strengthening of the impossibility results of both Dertouzos and Mok [9] and Hong and Leung [12].

## 4 Impossibility of Optimal Online Multiprocessor Scheduling for Sporadic and More General Task Systems

We now present the main result of this article. Our method of proving that optimal online algorithms do not exist for sporadic task systems is as follows.

1. Find a potentially feasible sporadic task system $\tau$ on some processing platform $\Pi$.
2. Prove that the task system is feasible a multiprocessor platform $\Pi$. This means that for any real-time instance generated by $\tau$ on $\Pi$ there exists a schedule on $\Pi$ that will meet all deadlines.
3. For the feasible task system $\tau$, show there exists a set of real-time instances generated by $\tau$ that are identical up to a time $t$ (denoted by $\mathcal{I}'(\tau)$); however, at time $t$ they require any online scheduling algorithm $\mathcal{A}$ to make a decision regarding which current jobs to schedule (i.e., there are more current jobs than processors at time $t$). Show that regardless of the choice made by $\mathcal{A}$ at time $t$, there exists a real-time instance in $\mathcal{I}'(\tau)$ that causes the choice made by $\mathcal{A}$ at time $t$ to result in a deadline miss.

In this brief section, we give the details of Steps 1 and 3. Step 3 especially gives insight into why optimal online scheduling of sporadic task systems is impossible. The proof of feasibility (Step 2), though very important to showing the nonexistence of optimal scheduling algorithms, is extremely complex and not necessary to understanding the main result of this paper; therefore, we have decided to defer the details of Step 2 until the next section (Section 5).

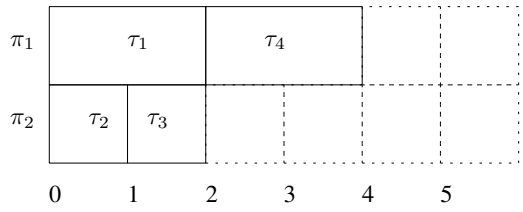
In accordance with Step 1 of the above approach, consider the following task system, $\tau^{\text{example}}$, comprised of six tasks (recall that a sporadic task is specified by three-tuple $(e_i, d_i, p_i)$) and described by Figure 1a.

**Theorem 2** *$\tau^{\text{example}}$ is feasible on two processors.*

**Proof:** Proved in Section 5. ■

| | $e_i$ | $d_i$ | $p_i$ |
|---|---|---|---|
| $\tau_1$ | 2 | 2 | 5 |
| $\tau_2$ | 1 | 1 | 5 |
| $\tau_3$ | 1 | 2 | 6 |
| $\tau_4$ | 2 | 4 | 100 |
| $\tau_5$ | 2 | 6 | 100 |
| $\tau_6$ | 4 | 8 | 100 |

(a) Task system $\tau^{\text{example}}$




(b) The times at which tasks $\tau_1$, $\tau_2$, $\tau_3$, and $\tau_4$ must execute

**Figure 1. System $\tau^{\text{example}}$ and Its Execution.**

**Lemma 2** *No optimal online algorithm exists for the multiprocessor scheduling real-time, constrained-deadline sporadic task systems on two processors.*

**Proof:** The proof is by contradiction. Assume there exists an optimal online algorithm, $\mathcal{A}$, for scheduling constrained-deadline sporadic real-time tasks on two processors. Then, by Theorem 2, $\mathcal{A}$ must find a valid schedule for $\tau^{\text{example}}$ where no deadline is missed; more formally, for all $I \in \mathcal{I}^{\text{S}}(\tau^{\text{example}})$, the schedule $\mathcal{A}(I)$ is valid (Definition 6). Figure 1a shows task system $\tau^{\text{example}}$.

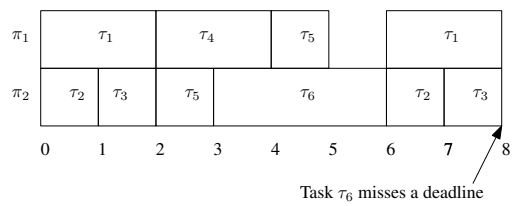
Let each task of $\tau^{\text{example}}$ release a job at time zero. Figure 1b shows the slots at which $\mathcal{A}$ must execute $\tau_1$, $\tau_2$, $\tau_3$, and $\tau_4$ (i.e., any other order would result in a deadline miss). Let $\mathcal{I}_{\text{zero}}(\tau^{\text{example}})$ be the set of all real-time instances generated by $\tau^{\text{example}}$ where each task generates a job at time instant zero and all jobs execute for their respective task's worst-case execution requirement; all real-time instances in $\mathcal{I}_{\text{zero}}(\tau^{\text{example}})$ must include the following six jobs (recall a real-time job is specified by $(A_i, E_i, D_i)$): $(0, 2, 2)$, $(0, 1, 1)$, $(0, 1, 2)$, $(0, 2, 4)$, $(0, 2, 6)$, and $(0, 4, 8)$. Note, that by the minimum separation parameter (period) of each task, the earliest the second job of any task may be generated is at time five. So, for all $I$ and $I'$ in $\mathcal{I}_{\text{zero}}(\tau^{\text{example}})$, $I_{\leq 5}$ and $I'_{\leq 5}$ are identical.

For any $I \in \mathcal{I}_{\text{zero}}(\tau^{\text{example}})$, there exist two possible choices that $\mathcal{A}$ must make regarding the execution of $\tau_5$.

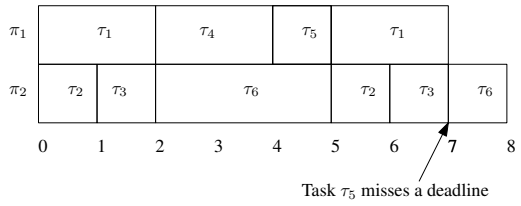
1. $\mathcal{A}$ schedules $\tau_5$ for $x$ $(0 < x \leq 2)$ units of time in the interval $(2, 4]$.
2. $\mathcal{A}$ does not schedule $\tau_5$ in the interval $(2, 4]$.

Since $\mathcal{A}$ is an online scheduling algorithm, by Definition 3, any $I, I'$ $\in \mathcal{I}_{\text{zero}}(\tau^{\text{example}})$ where $I_{\leq 5} = I'_{\leq 5}$ implies that the schedule generated by $\mathcal{A}$ for both $I$ and $I'$ is identical up to $t = 5$. Thus, algorithm $\mathcal{A}$ will make the same choice (either choice 1 or 2, above) for all instances in $\mathcal{I}_{\text{zero}}(\tau^{\text{example}})$. We will show that for either choice made by algorithm $\mathcal{A}$ there exists an $I_{\text{miss}} \in \mathcal{I}_{\text{zero}}(\tau^{\text{example}})$ that forces a deadline miss. Let us consider both cases.

1. *$\mathcal{A}$ schedules $\tau_5$ for $x$ $(0 < x \leq 2)$ units of time in the interval* $(2, 4]$: Consider any real-time instance $I$ in $\mathcal{I}_{\text{zero}}(\tau^{\text{example}})$ where, in addition to the six jobs that all real-time instances in $\mathcal{I}_{\text{zero}}(\tau^{\text{example}})$ must contain, $I$ includes a job generated by $\tau_1$, $\tau_2$, and $\tau_3$ at $t = 6$; that is, $I$ must include the jobs: $(6, 2, 2)$, $(6, 1, 1)$, and $(6, 1, 2)$. It is obvious that the two processors are fully utilized by $\tau_1$, $\tau_2$, and $\tau_3$ over the interval $(6, 8]$; therefore, $\tau_6$ may not execute over the interval $(6, 8]$ (otherwise, either $\tau_1$, $\tau_2$, or $\tau_3$ will miss a deadline). This implies that $\tau_6$ must execute in the interval $(2, 6]$ given real-time instance $I$. However, $\mathcal{I}$ chose to execute $\tau_5$ in $(2, 4]$ for $x$ time units, and $\tau_4$ requires a processor to execute job $(0, 2, 4)$ continuously. Thus, given the choice by $\mathcal{A}$ and real-time instance $I$, there only exists $4 - x$ units of time in which $\tau_6$ may execute in the interval $(2, 4]$; $\tau_6$ will miss a deadline at $t = 8$. Figure 2a shows this scenario.




(a) $\mathcal{A}$ schedules $\tau_5$ for $x$ $(0 < x \leq 2)$ units of time in the interval $(2, 4[$.




(b) $\mathcal{A}$ does not schedule $\tau_5$ in the interval $(2, 4[$

**Figure 2. Two Execution Scenarios for $\tau^{\text{example}}$.**

2. *$\mathcal{A}$ does not schedule $\tau_5$ in the interval* $(2, 4]$: Consider any real-time instance $I'$ in $\mathcal{I}_{\text{zero}}(\tau^{\text{example}})$ where, in addition to the six jobs that all real-time instances in $\mathcal{I}_{\text{zero}}(\tau^{\text{example}})$ must contain, $I'$ includes a job generated by $\tau_1$ and $\tau_2$ at $t = 5$; that is, $I'$ must include the jobs $(5, 2, 2)$ and $5, 1, 1)$. It is clear that the two processors are fully utilized by $\tau_1$ and $\tau_2$ over interval $(5, 6]$. However, since $\mathcal{A}$ chose not to execute $\tau_5$ in the interval $(2, 4]$, $\tau_5$ must continuously execute in the interval $(4, 8]$ to meet its deadline. In this scenario, three jobs must continuously execute in the interval $(5, 6]$. Therefore, either $\tau_1$, $\tau_2$, or $\tau_5$ will miss a deadline in the interval $(5, 6]$. Figure 2b illustrates this scenario.

Since for any of the choices made by $\mathcal{A}$ over the interval $(2, 4]$, there exists a real-time instance $I \in \mathcal{I}_{\text{zero}}(\tau^{\text{example}})$ that causes $\mathcal{A}$ to miss a deadline, this contradicts our assumption that there exists an optimal algorithm $\mathcal{A}$. Therefore, no optimal algorithm for scheduling sporadic real-time tasks upon a two-processor platform can exist.

■

We may easily generalize the above lemma to an arbitrary number of processors ($m > 1$).

**Theorem 3** *No optimal online algorithm exists for the multiprocessor scheduling real-time, constrained-deadline sporadic task systems on two or more processors.*

**Proof:** For any $\Pi$ comprised of $m > 1$ identical unit-speed processors, consider the task system $\tau' \stackrel{\text{def}}{=} \tau^{\text{example}} \cup \{\tau'_1, \tau'_2, \ldots, \tau'_{m-2}\}$ where $\tau'_i = (1, 1, 1)$ for all $0 < i \leq m - 2$. It is easy to see that $\tau'$ is feasible on $\Pi$, as we can dedicate a processor to each of the tasks in $\{\tau'_1, \tau'_2, \ldots, \tau'_{m-2}\}$ and by Theorem 2 $\tau^{\text{example}}$ is feasible on the remaining two processors. The argument of Lemma 2 holds in the case where each of $\{\tau'_1, \tau'_2, \ldots, \tau'_{m-2}\}$ generate jobs at time zero and successive jobs as soon as legally allowable. Therefore, the jobs generated by $\tau^{\text{example}}$ cannot use the additional processors, and the argument of the lemma is identical. ■

The above negative result immediately extends to any task model that generalizes the sporadic task model. The reason is that for any model M that generalizes the sporadic model, there exists a $\tau'^{\text{M}}$ specified in model M such that $I \in \mathcal{I}^{\text{M}}(\tau'^{\text{M}})$ if and only if $I \in \mathcal{I}^{\text{S}}(\tau')$. Therefore, the argument of Lemma 2 is unchanged for this more general task system (e.g., arbitrary-deadline sporadic task systems or GMF task systems).

**Corollary 1** *No optimal online algorithm exists on two or more processors for the multiprocessor scheduling of real-time task systems in models that generalize the sporadic task model.*

## 5 Feasibility of Sporadic Task System $\tau^{\text{example}}$ on Two Processors

Section 4 introduced task system $\tau^{\text{example}}$ (given by Figure 1a) that is used to prove that optimal online multiprocessor scheduling of arbitrary and constrained task systems is impossible. In this section, we give a formal proof of Theorem 2; that is, task system $\tau^{\text{example}}$ is feasible on two processors.

In Section 5.1, we informally outline our proof. In Section 5.2, we introduce additional formal notation involved in $\tau^{\text{example}}$'s feasibility proof. In Section 5.3, we give the entire feasibility proof.

### 5.1 Outline

The goal of Theorem 2 is to show that task system $\tau^{\text{example}}$ is feasible on two processors. However, we are unaware of any existing, non-trivial, exact feasibility test for constrained-deadline task systems on a multiprocessor platform that could precisely determine whether $\tau^{\text{example}}$ is feasible on two processors or not. For instance, the task system does not satisfy the sufficient feasibility condition [11]. The sufficient conditions for feasibility of sporadic task systems of Baker and Cirinei [2] only apply to single processors. Finally, the exact "brute-force" multiprocessor schedulability algorithm of Baker and Cirinei [3] does not trivially extend to multiprocessor feasibility. Furthermore, even if one could extend the brute-force result to multiprocessor feasibility, our approach does not assume integer arrival times and execution (as would be required by the current brute-force approach). Thus, since we may not validate the feasibility of $\tau^{\text{example}}$ with previously-known techniques, we must tailor an argument specially for task system $\tau^{\text{example}}$. Specifically, we must show that for every legal real-time instance $I$ generated by task system $\tau^{\text{example}}$, there exists a valid schedule in which no deadlines are missed (i.e., $\tau^{\text{example}}$ satisfies the definition of feasible task system according to Definition 8).

The approach that we take for proving Theorem 2 is to show, for any $I \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau^{\text{example}})$, that a valid schedule may be constructed for $I$ on two processors[5]. It turns out that it is very easy to find a schedule on two processors for the set of tasks $\tau^{\text{example}} \setminus \{\tau_6\}$; so, we construct this schedule, denoted $S_I$, for the jobs of $\tau^{\text{example}} \setminus \{\tau_6\}$ in real-time instance $I$. If the jobs of $\tau_6$ in instance $I$ can execute completely during the processor idle times for $S_I$ (i.e., when jobs of $\tau^{\text{example}} \setminus \{\tau_6\}$ are not executing in $S_I$), then we have shown that a valid schedule exists for instance $I$. However, it is possible that there does not exist sufficient idle processor time to execute every job of $\tau_6$ in $S_I$. Therefore, we may need to modify schedule $S_I$ further. Our approach considers up to two additional modified schedules, $S'_I$ and $S''_I$ – defined separately for ease of presentation and clarity. Our final step is to show that if $\tau_6$ could not complete in either $S_I$ or $S'_I$, all jobs of $\tau_6$ must complete in $S''_I$. The following steps informally explain our proof of showing that a valid schedule exists on two processors for any $I \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau^{\text{example}})$. Figure 3 gives a diagram of the steps of the proof.

**Step 0)** *Partition $\tau^{\text{example}} \setminus \{\tau_6\}$:* Consider a partition of $\tau^{\text{example}} \setminus \{\tau_6\}$ into two sets:

$$\tau^A \stackrel{\text{def}}{=} \{\tau_1, \tau_4\}, \tag{13}$$

and

$$\tau^B \stackrel{\text{def}}{=} \{\tau_2, \tau_3, \tau_5\}. \tag{14}$$

**Step 1)** *Construct schedule $S_I$ to show that $\tau^{\text{example}} \setminus \{\tau_6\}$ is feasible on two processor:* Using known uniprocessor scheduling algorithms, we show that $\tau^A$ may be correctly scheduled on processor $\pi_1$ and $\tau^B$ may be scheduled on processor $\pi_2$.

**Step 2)** *Construct a modified schedule $S'_I$:* If the jobs of $\tau_6$ cannot completely execute by their deadlines on processor $\pi_2$ (the less "loaded" of the two processors in $S_I$) during the idle time instants in schedule $S_I$, we will construct a new schedule $S'_I$. For any real-time instance $I$, $S'_I$ is a *global* schedule (i.e., non-partitioned) constructed by moving as much work as possible to the first processor $\pi_1$ (with respect to idle times in $S_I$ schedule for processor $\pi_1$).

[5]Please note that we only consider real-time instances in $I \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau^{\text{example}})$; the feasibility of any instance $I' \in \mathcal{I}^{\text{S}}(\tau^{\text{example}})$ follows from the fact that there exists an $I \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau^{\text{example}})$ such that $I' \in \mathcal{F}(I)$. So, we only need to consider a valid schedule $S''_I$ and it suffices to use the same schedule for $I'$ (except the jobs of $I'$ will potentially execute for less than the jobs of $I$).

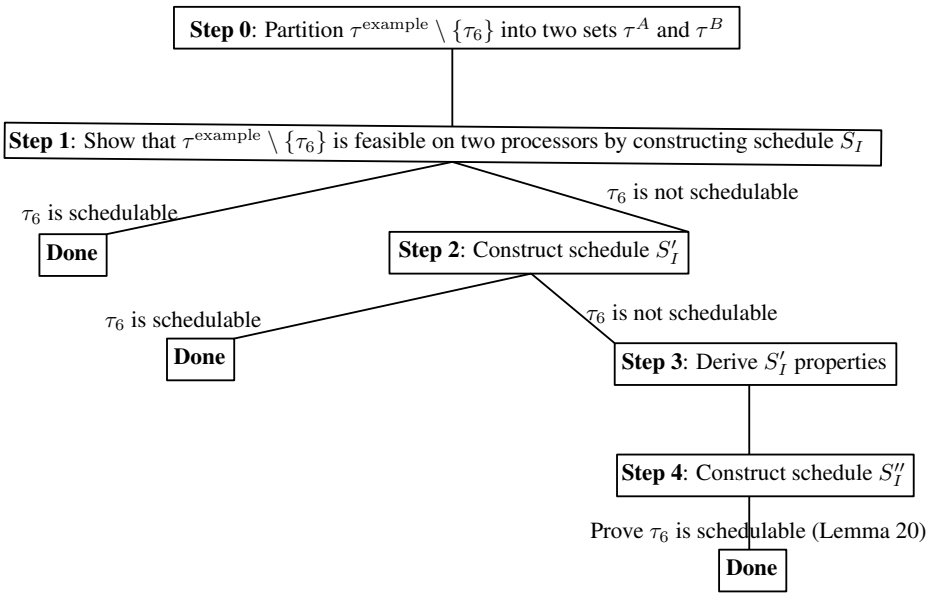



**Figure 3. Logical steps in proof of Theorem 2.**

**Step 3)** *Derive properties of schedule $S'_I$ if $\tau_6$ cannot complete execution:* We will derive several properties in the event that $\tau_6$ cannot complete during the idle instants in schedule $S'_I$. These properties will be useful in defining a second modified schedule $S''_I$ in which $\tau_6$ can complete execution.

**Step 4)** *Construct a second modified schedule $S''_I$ that leaves sufficient room for $\tau_6$ to be completely assigned to the second processor*: Again, if $\tau_6$ cannot completely execute during the idle times instants on processor $\pi_2$ in schedule $S'_I$, we construct a second modified schedule $S''_I$. The properties of the previous step will be used to show that a schedule $S''_I$ can always be constructed that leaves the second processor idle for four units between the release and deadline of a any job of $\tau_6$. Obviously, $\tau_6$ can be completely assigned to these idle times. Therefore, $\tau^{\text{example}}$ is feasible on two unit-capacity processors (Theorem 2).

In the next section, we discuss some additional notation needed for our proof. In Section 5.3, we formally carry-out the steps outlined in this subsection.

### 5.2 Notation

In this section, we present general notation for describing the scheduling and behavior of a sporadic task system $\tau$. The remainder of this section heavily relies on the notation presented in Sections 2.4 and 2.5. The notation defined for the remainder of this section will assume that $\tau$ is a constrained-deadline system (i.e., for all $\tau_i \in \tau$, $d_i \leq p_i$). For each $I \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau)$, let $I(\tau_i) \subseteq I$ denote the jobs generated by $\tau_i$ in instance $I$.

The next two functions give the "nearest" job release-time and deadline with respect to a given time $t$ and real-time instance $I(\tau_i)$.

**Definition 9 (Job-Release Function)** *If $\tau_i$ is current at time $t$ in real-time instance $I$ then $r_i(I,t)$ is the release time of the most recently released job of $\tau_i$ (with respect to time $t$). Otherwise, $r_i(I,t) = \infty$ if $\tau_i$ is not in a scheduling window at time $t$. More formally,*

$$r_i(I,t) \stackrel{def}{=} \begin{cases} A_k\,, & \text{if } \exists J_k \in I(\tau_i) \text{ such that } A_k \leq t \leq A_k + D_k \\ \infty\,, & \text{otherwise.} \end{cases} \tag{15}$$

**Definition 10 (Job-Deadline Function)** *If $\tau_i$ is current at time $t$ for real-time instance $I$ then $d_i(I,t)$ is the absolute deadline of the most recently released job of $\tau_i$ (with respect to time $t$). Otherwise, $d_i(I,t) = -\infty$ if $\tau_i$ is not in a scheduling window at time $t$.*

$$d_i(I,t) \stackrel{def}{=} \begin{cases} A_k + D_k\ , & \text{if } \exists J_k \in I(\tau_i) \text{ such that } A_k \leq t \leq A_k + D_k \\ -\infty\ , & \text{otherwise.} \end{cases} \tag{16}$$

The following function is useful for identifying the current current job (if any) of task $\tau_i$ at time $t$.

**Definition 11 (Current-Job Function)** *If $\tau_i$ is current at time $t$ for real-time instance $I$, $\varphi_i(I,t)$ is the current job at time $t$. Otherwise, $\varphi_i(I,t) = \perp$, if $\tau_i$ is not in a scheduling window at time $t$.*

$$\varphi_i(I,t) \stackrel{def}{=} \begin{cases} J_k\ , & \text{if } \exists J_k \in I(\tau_i) \text{ such that } A_k \leq t \leq A_k + D_k \\ \perp\ , & \text{otherwise.} \end{cases} \tag{17}$$

Similar to Definition 2 which defined a schedule function with respect to jobs of a real-time instance, we can define the schedule $S$ as a function with respect to task $\tau_i$.

**Definition 12 (Task-Schedule Function)** *$S_I(\pi_\ell, t, \tau_i)$ is an indicator function denoting whether task $\tau_i$ is scheduled at time $t$ for schedule $S_I$. In other words,*

$$S_I(\pi_\ell, t, \tau_i) \stackrel{def}{=} \begin{cases} 1\ , & \text{if } \exists J_k \in I(\tau_i) :: S_I(\pi_\ell, t, J_i) = 1 \\ 0\ , & \text{otherwise.} \end{cases} \tag{18}$$

The next definition defines the work that task $\tau_i$ receives on $\pi_\ell$ over a given interval. The job work function (Definition 4) is used.

**Definition 13 (Task-Work Function)** *$W_i(S_I, \pi_\ell, t_1, t_2)$ denotes the amount of processor time that $\tau_i$ receives from schedule $S_I$ on processor $\pi_\ell$ over the interval $[t_1, t_2)$ for real-time instance $I$. In other words,*

$$W_i(S_I, \pi_\ell, t_1, t_2) \stackrel{def}{=} \sum_{J_k \in I(\tau_i)} W(S_I, \pi_\ell, J_i, t_1, t_2). \tag{19}$$

**Definition 14 (Idle-Work Function)** *$W_\perp(S_I, \pi_\ell, t_1, t_2)$ denotes the total amount of processor time that schedule $S_I$ idles processor $\pi_\ell$ over the interval $[t_1, t_2)$ for real-time instance $I$. In other words,*

$$W_\perp(S_I, \pi_\ell, t_1, t_2) \stackrel{def}{=} W(S_I, \pi_\ell, \perp, t_1, t_2). \tag{20}$$

### 5.3 Proof

In this section, we prove Theorem 2 by following the steps outlined in Section 5.1. Obviously, Step 0 has already been given in the proof outline of Section 5.1; thus, we begin with Step 1. Section 5.3.1 gives the construction for schedule $S_I$ for Step 1. Section 5.3.2 describes the construction of schedule $S'_I$ for Step 2. Section 5.3.3 proves several important properties about $S'_I$, if $\tau_6$ cannot be scheduled during the idle times (Step 3). Finally, Section 5.3.4 defines schedule $S''_I$ which can be shown to accommodate all jobs of task $\tau_6$ on processor $\pi_2$ (Step 4).

### 5.3.1 Step 1: Construction of Schedule $S_I$

The first step of the outline (Section 5.1) of the proof requires us to show that the partition $\tau^A$ and $\tau^B$ of $\tau^{\text{example}} \setminus \{\tau_6\}$ is feasible on two processors and give a valid schedule for real-time instance $I \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau)$. We can easily obtain feasibility of this task system by partitioning $\tau^{\text{example}} \setminus \{\tau_6\}$ into two sets and scheduling each subset on its own processor using a uniprocessor scheduling algorithm called the *deadline-monotonic* (DM) scheduling algorithm. For each processor, DM executes at any time instant the active job of the task with the smallest relative deadline parameter (among the set of all tasks assigned to that processor with active jobs). For simplicity of analysis, we will use DM on each processor.

Audsley et al. [1] developed a test to determine whether each task in a *constrained-deadline* task system can be scheduled by DM on a single processor to always meet all deadlines. Let $\mathbf{T}_{H_i}$ be the set of tasks with priority greater than or equal to task $\tau_i$ under the DM priority assignment. The following theorem restates their result.

**Theorem 4 (from [1])** *In a constrained-deadline, sporadic task system, task $\tau_i$ always meets all deadlines using* DM *on a preemptive uniprocessor if and only if $\exists t \in (0, d_i]$ such that*

$$\left( \text{CBF}(\tau_i, t) \stackrel{def}{=} \sum_{\tau_j \in \mathbf{T}_{H_i}} \text{RBF}(\tau_j, t) + e_i \right) \leq t. \tag{21}$$

■

Using this result, we obtain the following lemma which states that $\tau^{\text{example}} \setminus \{\tau_6\}$ is feasible on the given two-processor platform:

**Lemma 3** *$\tau^{\text{example}} \setminus \{\tau_6\}$ is feasible on a multiprocessor platform composed of two unit-capacity processors.*

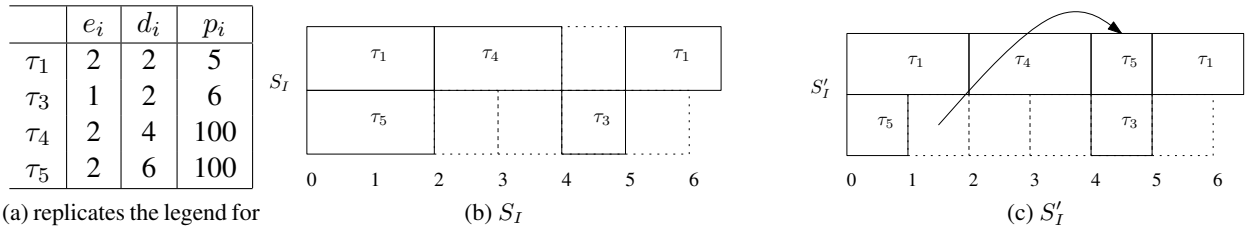
**Proof:** For partition $\tau^A$ and $\tau^B$ of $\tau^{\text{example}} \setminus \{\tau_6\}$ (Equations 13 and 14), assign $\tau^A$ to $\pi_1$ and $\tau^B$ to $\pi_2$. It is easy to verify by Theorem 4 that $\tau^A$ and $\tau^B$ are feasible with respect to their assigned processors. First, we will show that $\tau^A$ is feasible on processor $\pi_1$. $\tau_1$ always meets all deadlines (according to Theorem 4) on $\pi_1$ with respect to task system $\tau^A$ because $\text{CBF}(\tau_1, 2) = 2 \leq 2$. Similarly, $\tau_4$ always meets all deadlines on $\pi_1$ because $\text{CBF}(\tau_4, 4) = 2+2 = 4 \leq 4$. Since both of these tasks always meet all deadlines using DM on $\pi_1$ over all real-time instance $I_A \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau^A)$, $\tau^A$ is feasible on $\pi_1$ according to Definition 8.

Next, we will show that $\tau^B$ is feasible on processor $\pi_2$. $\tau_2$ always meets all deadlines on $\pi_2$ because $\text{CBF}(\tau_2, 1) = 1 \leq 1$. $\tau_3$ always meets all deadlines on $\pi_2$ due to $\text{CBF}(\tau_3, 2) = 1 + 1 = 2 \leq 2$. Finally, $\tau_5$ always meets all deadlines on $\pi$ because $\text{CBF}(\tau_5, 4) = 1 + 1 + 2 = 4 \leq 4$. Since all three of these tasks always meet all deadlines using DM on $\pi_2$ over all real-time instances $I_B \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau^B)$, $\tau^B$ is feasible on $\pi_2$ according to Definition 8. Combining the two uniprocessor schedules from DM, we get a valid multiprocessor schedule for $\tau^{\text{example}} \setminus \{\tau_6\}$, and the lemma follows. ■

Let $S_I$ be the schedule constructed by DM on each processor for task system $\tau^{\text{example}} \setminus \{\tau_6\}$ with partitions $\tau^A$ and $\tau^B$. From the previous argument, $S_I$ is valid for $I$ (with $\tau_6$'s jobs excluded).

### 5.3.2 Step 2: Construction of Schedule $S'_I$

If the jobs generated by $\tau_6$ in real-time instance $I$ cannot complete by their deadlines in the idle times of $S_I$, we must proceed to Step 2 of our proof outline: construct a schedule $S'_I$ that is globally (non-partitioned) feasible. The goal of this step is to move as much computation off processor $\pi_2$ as possible. To accomplish this goal, for every idle instant on processor $\pi_1$ in schedule $S_I$, we move a task in its scheduling window on $\pi_2$ to $\pi_1$ (if such a task exists). The construction "builds" schedule $S'_I$ for processor $\pi_1$, first. After $S'_I(\pi_1, t)$ is constructed, then $S'_I$




| | $e_i$ | $d_i$ | $p_i$ |
|---|---|---|---|
| $\tau_1$ | 2 | 2 | 5 |
| $\tau_3$ | 1 | 2 | 6 |
| $\tau_4$ | 2 | 4 | 100 |
| $\tau_5$ | 2 | 6 | 100 |

(a) replicates the legend for these tasks

(b) $S_I$

(c) $S'_I$

**Figure 4. Construction of Schedule $S'_I$. Note that the execution of $\tau_5$ in the interval $[1, 2)$ is moved from the second processor to $[4, 5)$ on the first processor.**

is constructed for $\pi_2$. Note that such a schedule could not be constructed online (i.e., it is an off-line constructed schedule), since $S'_I(\pi_2, t)$ may require that $S'_I(\pi_1, t')$ be known for some $t' > t$ (i.e., $S'_I(\pi_2, t)$ requires knowledge of future events). Constructing an offline schedule is not a contradiction of Theorem 3 as feasibility requires us only to construct (by any means) a valid schedule for any real-time instance that may be generated by $\tau^{\text{example}}$.

In schedule $S'_I(\pi_1, t)$, tasks of set $\tau^A$ (tasks $\tau_1$ and $\tau_4$) execute at exactly the same times as they did in schedule $S_I(\pi_1, t)$ (i.e., the uniprocessor rate-monotonic schedule for $\tau^A$ and $\tau^B$). However, the tasks of set $\tau^B$ move as much execution as possible (without disturbing tasks of $\tau^A$) from processor $\pi_2$ to processor $\pi_1$. Consider an arbitrary time $t$. $S'_I(\pi_1, t)$ is constructed using the following rules:

1. If at time $t$ processor $\pi_1$ is busy executing a job from tasks of $\tau^A$ in schedule $S_I$, then $S'_I(\pi_1, t)$ equals $S_I(\pi_1, t)$.
2. If processor $\pi_1$ is idle at time $t$ in schedule $S_I$, then:
   (a) If task $\tau_5$ is in its scheduling window (i.e., $r_5(I, t) < \infty$) and it has not already executed for more than $e_5$ time units in $S'_I$ on processor $\pi_1$, then $S'_I$ at time $t$ is set to the current job of $\tau_5$ – i.e $S'_I(\pi_1, t) = \varphi_5(I, t)$;
   (b) *else*, if task $\tau_2$ is in its scheduling window (i.e., $r_2(I, t) < \infty$) and it has not already executed for more than $e_2$ time units in $S'$ on processor $\pi_1$, then $S'_I$ at time $t$ is set to the current job of $\tau_2$ – i.e $S'_I(\pi_1, t) = \varphi_2(I, t)$;
   (c) *else*, if task $\tau_3$ is in its scheduling window (i.e., $r_3(I, t) < \infty$) and it has not already executed for more than $e_3$ time units in $S'$ on processor $\pi_1$, then $S'_I$ at time $t$ is set to the current job of $\tau_3$ – i.e $S'_I(\pi_1, t) = \varphi_3(I, t)$;
   (d) *else*, leave processor $\pi_1$ idle.

Note the above order that we move jobs of tasks (i.e., in order of $\tau_5$, $\tau_2$, and $\tau_3$) is significant.

The execution of jobs of tasks in $\tau^B$ that could not be moved to processor $\pi_1$ is executed on processor $\pi_2$ (with the added constraint that a task does not execute in parallel with itself). For arbitrary time $t$, $S'_I(\pi_2, t)$ is constructed using the following rule: if, at time instant $t$, a job $J_k$ of task $\tau_i \in \tau^B$ is executing on processor $\pi_2$ in schedule $S_I$ (i.e., $S_I(\pi_2, t, J_k) = 1$), then $J_k$ will also execute on processor $\pi_2$ at time instant $t$ in schedule $S'_I$ only if the following two conditions are true,

1. $J_k$ is not executing on processor $\pi_1$ at time $t$ in schedule $S'_I$ (i.e., $S'_I(\pi_2, t, J_k) = 0$), and

2. the total time that job $J_k$ has executed on processor $\pi_1$ between its arrival and its absolute deadline and on processor $\pi_2$ between its arrival and time $t$ in schedule $S'_I$ is strictly less than $e_i$.

Figure 4 presents a visual example comparing schedules $S_I$ and $S'_I$. The following construction is the inductive formal definition of the modified schedule for all $I \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau^{\mathrm{example}} \setminus \{\tau_6\})$ and $t \geq 0$. Please note that $S'_I(\pi_1, t)$ is inductively constructed first for all $t \geq 0$. $S'_I(\pi_2, t)$ is constructed after $S'_I$ for processor $\pi_1$. Also, note that $S'_I$ is not *work-conserving* in the sense that a processor may be idle at time $t$, even if there are active jobs with remaining execution.

$$S'_I(\pi_1,t) \stackrel{\text{def}}{=} \begin{cases} S_I(\pi_1,t)\,, & \text{if } S_I(\pi_1,t) \neq \bot, \\ \varphi_5(I,t)\,, & \text{if } r_5(I,t) < \infty \text{ and } W_5(S'_I,\pi_1,r_5(I,t),t) < e_5, \\ \varphi_2(I,t)\,, & \text{if } r_2(I,t) < \infty \text{ and } W_2(S'_I,\pi_1,r_2(I,t),t) < e_2, \\ \varphi_3(I,t)\,, & \text{if } r_3(I,t) < \infty \text{ and } W_3(S'_I,\pi_1,r_3(I,t),t) < e_3, \\ \bot\,, & \text{otherwise} \end{cases}$$

$$S'_I(\pi_2,t) \stackrel{\text{def}}{=} \begin{cases} \varphi_2(I,t)\,, & \text{if } (S_I(\pi_2,t,\tau_2) = 1) \text{ and } (S'_I(\pi_1,t,\tau_2) = 0) \text{ and} \\ & (W_2(S'_I,\pi_1,r_2(I,t),d_2(I,t)) + W_2(S'_I,\pi_2,r_2(I,t),t) < e_2), \\ \varphi_3(I,t)\,, & \text{if } (S_I(\pi_2,t,\tau_3) = 1) \text{ and } (S'_I(\pi_1,t,\tau_3) = 0) \text{ and} \\ & (W_3(S'_I,\pi_1,r_3(I,t),d_3(I,t)) + W_3(S'_I,\pi_2,r_3(I,t),t) < e_3), \\ \varphi_5(I,t)\,, & \text{if } (S_I(\pi_2,t,\tau_5) = 1) \text{ and } (S'_I(\pi_1,t,\tau_5) = 0) \text{ and} \\ & (W_5(S'_I,\pi_1,r_5(I,t),d_5(I,t)) + W_5(S'_I,\pi_2,r_5(I,t),t) < e_5), \\ \bot\,, & \text{otherwise} \end{cases} \tag{22}$$

**Lemma 4** *$S'_I$ is valid on $\Pi$ for any $I$ with respect to jobs of $\tau^{\mathrm{example}} \setminus \{\tau_6\}$.*

**Proof:** Schedule $S'_I$ is obviously valid for the jobs generated by $\tau^{\mathrm{example}} \setminus \{\tau_6\}$ in instance $I$. Each job, by definition of $S'_I$ above only executes within its scheduling window, does not execute concurrently with itself on both processors, and executes exactly for its execution requirement. ■

### 5.3.3 Step 3: Properties of Schedule $S'_I$

If $S'_I$ does not have sufficient idle time to schedule $\tau_6$ entirely on processor $\pi_2$, then there must exist a job $J_6 \in I$ that does not meet its deadline, with respect to the idle time in schedule $S'_I$. In this section, we prove several lemmas which characterize the properties of schedule $S'_I$ with respect to the $J_6$'s scheduling window. The main observation from these properties is that the jobs of $I$ are constrained in how their scheduling windows intersect, if $J_6$ cannot be scheduled in $S'_I$. We will exploit these intersection constraints on job in the next section (Step 4) when we define schedule $S''_I$.

Let $A_6$ be the arrival of $J_6$. Since the relative deadline of $\tau_6$ is $d_6 = 8$, the scheduling window of $J_6$ is $[A_6, A_6+8)$. We will start by making an observation on the maximum possible amount that jobs of $\tau^{\mathrm{example}} \setminus \{\tau_6\}$ could execute in any schedule over an interval of length six, eight, and ten. These observations will be useful to reason about the amount of work that could occur over the scheduling windows of jobs of $\tau_5$ and $\tau_6$.

**Observation 1** *Table 1 presents upper-bounds on $W_i(S,\pi_1,t,t+L) + W_i(S,\pi_2,t,t+L)$ for intervals of length $L \in \{6,8,10\}$ for any valid schedule $S$, task $\tau_i \in \tau^{\mathrm{example}} \setminus \{\tau_6\}$ and time-instant $t$. The upper bounds for $L = 8$ and $L = 10$ may easily be observed by noting that $\tau_1$, $\tau_2$, and $\tau_3$ have at most two jobs with scheduling windows that intersect with an interval $[t, t+L)$ for $8 \leq L \leq 10$, and $\tau_4$ and $\tau_5$ can have at most one job with scheduling window intersecting $[t, t+L)$. Similarly, for $L = 6$, at most two jobs of $\tau_1$, $\tau_2$, and $\tau_3$ and a single job of $\tau_4$ and $\tau_5$ may over lap with the interval $[t, t+6)$; however, the maximum intersection between the scheduling windows of $\tau_1$'s jobs and $[t, t+6)$ is three, due to the fact that $p_1$ equals five. We also point out that upper bounds on $W_i(S,\pi_1,t,t+L) + W_i(S,\pi_2,t,t+L)$ are monotonically non-decreasing with $L$.*

| Task | $\geq W_i(S, \pi_1, t, t+L) + W_i(S, \pi_2, t, t+L)$ | | |
|---|---|---|---|
| | $L = 6$ | $L = 8$ | $L = 10$ |
| $\tau_1$ | 3 | 4 | 4 |
| $\tau_2$ | 2 | 2 | 2 |
| $\tau_3$ | 2 | 2 | 2 |
| $\tau_4$ | 2 | 2 | 2 |
| $\tau_5$ | 2 | 2 | 2 |

**Table 1. Upper bounds on the execution of tasks over intervals $[t, t+L)$ for various values of $L$.**

The first property we show for $S'_I$ is in regard to the execution of jobs of $\tau_5$ over the $J_6$'s scheduling window of $[A_6, A_6 + 8)$. If there was not sufficient idle time in $S'_I$ to completely schedule $J_6$, a job of $\tau_5$ must have its scheduling window intersect with $[A_6, A_6 + 8)$. Furthermore, a job of $\tau_5$ must execute for a non-zero amount of time on processor $\pi_2$ over $[A_6, A_6 + 8)$. The following lemma formally shows this property.

**Lemma 5** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then there exists a single job $J_5 \in I$ of $\tau_5$ with scheduling window $[A_5, A_5 + 6)$ where*

$$[A_5, A_5 + 6) \cap [A_6, A_6 + 8) \neq \emptyset. \tag{23}$$

*Furthermore, let $\alpha$ equal the execution of $J_5$ on processor $\pi_2$ over $[A_5, A_5+6)$ (i.e., $\alpha \stackrel{def}{=} W_5(S'_I, \pi_2, A_5, A_5+6)$). It must be that*

$$\alpha > 0. \tag{24}$$

*and that $J_5$ executes for some non-zero amount of time $\leq \alpha$ on $\pi_2$ over $[A_6, A_6+8)$ (i.e., $W_5(S'_I, \pi_2, A_6, A_6+8) \leq \alpha$).*

**Proof:** First note, that since the period of $\tau_5$, $p_5$, equals 100, at most one job of $\tau_5$ could possibly have its scheduling window intersect with the interval $[A_6, A_6 + 8)$. We will now show (by contradiction) that exactly one job of $\tau_5$ intersects $J_6$'s scheduling window and executes during this interval on processor $\pi_2$. Assume the lemma is false: a job of $\tau_5$ does not execute on processor $\pi_2$ over $J_6$'s scheduling window in valid schedule $S'_I$. Then, exactly one of the following three cases is true:

**Case 1** there does not exist a job $J_5 \in I$ with $[A_5, A_5 + 6) \cap [A_6, A_6 + 8) \neq \emptyset$.

**Case 2** there exists a job $J_5 \in I$ of $\tau_5$ with $[A_5, A_5 + 6) \cap [A_6, A_6 + 8) \neq \emptyset$, but $J_5$ does not execute on processor $\pi_2$ over the interval $[A_5, A_5 + 6)$ (i.e., $\alpha = 0$); or,

**Case 3** there exists a job $J_5 \in I$ of $\tau_5$ with $[A_5, A_5 + 6) \cap [A_6, A_6 + 8) \neq \emptyset$ and $J_5$ executes on processor $\pi_2$ for $\alpha > 0$ over the interval $[A_5, A_5 + 6)$, but does not execute over $[A_6, A_6 + 8)$;

By construction of $S'_I$, the only other tasks of $\tau^{\text{example}} \setminus \{\tau_6\}$, other than $\tau_5$, that are executed on $\pi_2$ in $S'_I$ are $\tau_2$ or $\tau_3$. Since $J_6$'s execution requirement, $E_6$, is 4, the execution of $\tau_2$ and $\tau_3$ on processor $\pi_2$ in schedule $S'_I$ over the interval $[A_6, A_6 + 8)$ must exceed four for $J_6$ to be unable to execute completely on $\pi_2$. However, by Observation 1, the most that $\tau_2$ and $\tau_3$ could execute over $[A_6, A_6 + 8)$ is four. Thus, $\tau_5$ must have executed on $\pi_2$ over $[A_6, A_6 + 8)$ in $S'_I$ for some non-zero amount of time in order for $J_6$ not to complete which contradicts the assumption of Cases 1, 2, and 3; the lemma follows. ■

The next lemma gives an upper bound on the amount of time that $J_5$ can execute on processor $\pi_2$ in schedule $S'_I$.

**Lemma 6** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then*

$$\alpha \leq 1. \tag{25}$$

**Proof:** If there is insufficient time in $S'_I$ to execute $J_6$, Lemma 5 states that a unique job $J_5$ of task $\tau_5$ must exist with a scheduling window $[A_5, A_5 + 6)$ that intersects $[A_6, A_6 + 8)$. Observation 1 implies that an upper bound on the execution of jobs of $\tau^A$ in $S'_I$ over the interval $[A_5, A_5 + 6)$ is at most five. Thus, the total amount of time that processor $\pi_1$ is idle over $[A_5, A_5 + 6)$ in the original schedule $S_I$ is at least one. $J_5$ executes on processor $\pi_1$ at most $2 - \alpha$, by Lemma 5. Assume that $\alpha > 1$. Then, $J_5$ executes on the processor $\pi_1$ for strictly less than one time unit in schedule $S'_I$. Thus, there exists $t \in [A_5, A_5 + 6)$ where $S'_I$ is executing a job not in task $\tau^A \cup \{\tau_5\}$ on processor $\pi_1$. However, the existence of execution of $J_5$ on processor $\pi_2$ contradicts the construction of $S'_I$ which moves as much of $J_5$'s to $\pi_1$ at instances whenever jobs of $\tau^A$ are not executing. Thus, our assumption that $\alpha > 1$ must be false and the lemma follows. ■

The next two lemmas (Lemmas 7 and 8) exactly characterize the jobs of $\tau_3$ and $\tau_2$ that must execute over $J_6$'s scheduling window.

**Lemma 7** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then there exists exactly two jobs of $\tau_3$, $J_3^1, J_3^2 \in I$ (where $A_3^1 + 6 \leq A_3^2$), such that*

$$\left([A_3^1, A_3^1 + 2) \cap [A_6, A_6 + 8) \neq \emptyset\right) \bigwedge \left([A_3^2, A_3^2 + 2) \cap [A_6, A_6 + 8) \neq \emptyset\right). \tag{26}$$

*Furthermore, both $J_3^1$ and $J_3^2$ execute for strictly more than $2 - \alpha$ time units on $\pi_2$ in $S'_I$ over $[A_6, A_6 + 8)$ (i.e., $W_3(S'_I, \pi_2, A_6, A_6 + 8) > 2 - \alpha$).*

**Proof:** Since $J_6$ cannot complete during the idle times in $S'_I$, the execution on processor $\pi_2$ over the $J_6$'s scheduling window $[A_6, A_6 + 8)$ must exceed four time units; otherwise, $J_6$ could complete entirely on processor $\pi_2$. By definition of $S'_I$, only jobs of $\tau_2$, $\tau_3$, and $\tau_5$ execute on processor $\pi_2$. Observation 1 implies that $\tau_2$ can execute for at most two time units over $[A_6, A_6 + 8)$. By Lemma 5 and Lemma 6, $J_5$ executes for amount of time at most $\alpha \leq 1$ time units over $J_6$'s scheduling window on processor $\pi_2$. Thus, $\tau_3$ must execute for strictly more than $2 - \alpha$ time unit over $[A_6, A_6 + 8)$ on $\pi_2$ in $S'_I$. Since the execution requirement $e_3$ is one, there must be at least two jobs of $\tau_3$ that execute during $[A_6, A_6 + 8)$. The period and relative deadline parameter of $\tau_3$ ($p_3 = 6$ and $d_3 = 2$) imply that at most two jobs of $\tau_3$ can execute in $[A_6, A_6 + 8)$. Let $J_3^1$ and $J_3^2$ be the jobs of $\tau_3$ that execute in $[A_6, A_6 + 8)$ where $A_3^2 - A_3^1 \geq 6$ (by the period parameter). The fact that $J_3^1$ and $J_3^2$'s scheduling windows overlap with $J_6$'s scheduling window implies Equation 26. ■

**Lemma 8** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then there exists exactly two jobs of $\tau_2$, $J_2^1, J_2^2 \in I$ (where $A_2^1 + 5 \leq A_2^2$), such that*

$$\left([A_2^1, A_2^1 + 1) \cap [A_6, A_6 + 8) \neq \emptyset\right) \bigwedge \left([A_2^2, A_2^2 + 1) \cap [A_6, A_6 + 8) \neq \emptyset\right). \tag{27}$$

*Furthermore, both $J_2^1$ and $J_2^2$ execute for strictly more than $2 - \alpha$ time units on $\pi_2$ in $S'_I$ over $[A_6, A_6 + 8)$ (i.e., $W_2(S'_I, \pi_2, A_6, A_6 + 8) > 2 - \alpha$).*

**Proof:** Since $J_6$ cannot complete during the idle times in $S'_I$, the execution on processor $\pi_2$ over the $J_6$'s scheduling window $[A_6, A_6 + 8)$ must exceed four time units; otherwise, $J_6$ could complete entirely on processor $\pi_2$. By definition of $S'_I$, only jobs of $\tau_2$, $\tau_3$, and $\tau_5$ execute on processor $\pi_2$. Observation 1 implies that $\tau_2$ can execute for at most two time units over $[A_6, A_6 + 8)$. By Lemma 5 and Lemma 6, $J_5$ executes for amount of time at most $\alpha \leq 1$ time units over $J_6$'s scheduling window on processor $\pi_2$. Thus, $\tau_2$ must execute for strictly more

than $2-\alpha$ time unit over $[A_6, A_6+8)$ on $\pi_2$ in $S'_I$. Since the execution requirement $e_2$ is one, there must be at least two jobs of $\tau_2$ that execute during $[A_6, A_6+8)$. The period and relative deadline parameter of $\tau_1$ ($p_1 = 5$ and $d_1 = 1$) imply that at most two jobs of $\tau_1$ can execute in $[A_6, A_6+8)$. Let $J_2^1$ and $J_2^2$ be the jobs of $\tau_2$ that execute in $[A_6, A_6+8)$ where $A_2^2 - A_2^1 \geq 5$ (by the period parameter). The fact that $J_2^1$ and $J_2^2$'s scheduling windows overlap with $J_6$'s scheduling window implies Equation 27. ■

The following corollary of Lemmas 6 and 8, showing that both $J_2^1$ and $J_2^2$ must execute on $\pi_2$ over $[A_6, A_6+8)$, will be useful in later proofs.

**Corollary 2** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6+8)$ to completely execute $J_6$, then both $J_2^1$ and $J_2^2$ execute for non-zero amounts of time on processor $\pi_2$ in the interval $[A_6, A_6+8)$.*

**Proof:** Lemma 8 states that $J_2^1$ and $J_2^2$ together must execute for strictly greater than $2-\alpha$ time on processor $\pi_2$ over the interval $[A_6, A_6+8)$. Lemma 6 show that $\alpha \leq 1$; thus, the execution of both jobs over interval $[A_6, A_6+8)$ must exceed one. Since $e_2 = 1$, both $J_2^1$ and $J_2^2$ must execute for non-zero amounts of time in $[A_6, A_6+8)$. ■

The previous two lemmas and corollary gave a lower bound on the execution of jobs of either $\tau_2$ or $\tau_3$ over the interval $[A_6, A_6+8)$ on processor $\pi_2$. In the next lemma, we derive a lower bound on the combined execution of $\tau_2$ and $\tau_3$ over this same interval and processor.

**Lemma 9** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6+8)$ to completely execute $J_6$, then $\tau_2$ and $\tau_3$ execute on processor $\pi_2$ over the interval $[A_6, A_6+8)$ for strictly more than $4-\alpha$ time units in $S'_I$. I.e.,*

$$W_2(S'_I, \pi_2, A_6, A_6+8) + W_3(S'_I, \pi_2, A_6, A_6+8) > 4 - \alpha \tag{28}$$

**Proof:** Since $J_6$ cannot complete during the idle times in $S'_I$, the execution on processor $\pi_2$ by jobs of $\tau^B$ over $J_6$'s scheduling window $[A_6, A_6+8)$ must exceed four units. Lemma 5 showed that the most $J_5$ could execute on processor $\pi_2$ over $[A_6, A_6+8)$ is $\alpha$. Thus, jobs of $\tau_2$ and $\tau_3$ must execute for strictly more than $4-\alpha$ time units on processor $\pi_2$ over $[A_6, A_6+8)$. ■

We now focus on jobs of tasks in $\tau^A$ whose scheduling windows overlap with $J_5$'s scheduling window. The above lemmas (Lemmas 5, 7, and 8) showed that a jobs of $\tau^B$ must have prevented $J_6$ from completing execution entirely on processor $\pi_2$. We follow this reasoning and show that a jobs of $\tau^A$ must have prevented $\tau_5$'s job, $J_5$ from completing its execution entirely on processor $\pi_1$. The next two properties of $S'_I$ show that exactly one job of $\tau_4$ executes in the scheduling window $[A_5, A_5+6)$ (Lemma 10) and exactly two jobs of $\tau_1$ execute in $[A_5, A_5+6)$ (Lemma 11).

**Lemma 10** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6+8)$ to completely execute $J_6$, then there exists a single job $J_4 \in I$ of $\tau_4$ such that*

$$[A_4, A_4+4) \cap [A_5, A_5+6) \neq \emptyset. \tag{29}$$

*Furthermore, $J_4$ executes for at least $1+\alpha$ units of time on $\pi_1$ in $S'_I$ over $[A_5, A_5+6)$ (i.e., $W_5(S'_I, \pi_1, A_5, A_5+6) > 0$).*

**Proof:** By Lemma 5, $J_5$ executes on processor $\pi_2$ for some $\alpha > 0$ amount of time in schedule $S'_I$. By construction, $S'_I$ executes $J_5$ at any time instant $t \in [A_5, A_5+6)$ where processor $\pi_1$ was idle in the original schedule $S_I$ (i.e., neither $\tau_1$ or $\tau_4$ were executing at time $t$). Since $J_5$ could only execute $2-\alpha$ ($\leq 1$) units on processor $\pi_1$ over its scheduling window, this implies that the total amount jobs of $\tau_1$ and $\tau_4$ execute over $[A_5, A_5+6)$ is exactly $4+\alpha$. By Observation 1, the most that jobs of $\tau_1$ could execute in this scheduling window is three time units; thus, there must exist at least one job $J_4 \in I$ such that $[A_4, A_4+4) \cap [A_5, A_5+6) \neq \emptyset$ where $J_4$ executes for at least $1+\alpha$ units on processor $\pi_1$ over $[A_5, A_5+6)$ in schedule $S'_I$. Since $\tau_4$'s period, $p_4$, equals 100, $J_4$ is unique. ■

**Lemma 11** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then there exists exactly two jobs of $\tau_1$, $J_1^1, J_1^2 \in I$ (where $A_1^1 + 5 \leq A_1^2$), such that*

$$\left([A_1^1, A_1^1 + 2) \cap [A_5, A_5 + 6) \neq \emptyset\right) \bigwedge \left([A_1^2, A_1^2 + 2) \cap [A_5, A_5 + 6) \neq \emptyset\right). \tag{30}$$

*Furthermore, the total execution of $J_1^1$ and $J_1^2$ must be at least $2+\alpha$ units of time on $\pi_1$ in $S'_I$ over $[A_5, A_5+6)$ (i.e., $W_1(S'_I, \pi_1, A_5, A_5 + 6) > 2 + \alpha$).*

**Proof:** Again, by Lemma 5, $J_5$ executes on $\pi_2$ for $\alpha$ time in $S'_I$. By identical reasoning as the proof for Lemma 10, $\tau_1$ and $\tau_4$ must execute for exactly $4 + \alpha$ units over the interval $[A_5, A_5 + 6)$. By Observation 1, the most that $\tau_4$ could execute in $J_5$'s scheduling window is two time units. Thus, jobs of $\tau_1$ must execute for at least $2 + \alpha$ time units over $J_5$'s scheduling window. Since the execution requirement of a single job of $\tau_1$ is one time unit, this implies there must exist at least two jobs $J_1^1, J_1^2 \in I$ of $\tau_1$ such that $\left([A_1^1, A_1^1 + 2) \cap [A_5, A_5 + 6) \neq \emptyset\right)$ and $\left([A_1^2, A_1^2 + 2) \cap [A_5, A_5 + 6) \neq \emptyset\right)$ that execute in $S'_I$ over $[A_5, A_5 + 6)$ on processor $\pi_1$ for more than two units of time. Assume that the arrival of $J_1^1$ precedes $J_1^2$. The period of $\tau_1$ ($p_1 = 5$) implies that $A_1^1 + 5 \leq A_1^2$ and that no more than two jobs of $\tau_1$ could execute in $[A_5, A_5 + 6)$. ■

We now focus our attention on the execution of jobs of $\tau^A \cup \{\tau_5\}$ that could prevent execution of $\tau_2$ and $\tau_3$ from being moved from processor $\pi_2$ to $\pi_1$. The next lemma (Lemma 12) shows that the scheduling window $[A_5, A_5+6)$ is a *continuously busy interval on processor* $\pi_1$ with respect to schedule $S'_I$ tasks $\tau^A \cup \{\tau_5\}$. A continuously busy interval for a processor with respect to a given collection of tasks and schedule is an interval $[t_1, t_2)$ where a job of the given task collection is executing in the schedule on the processor for all time $t \in [t_1, t_2)$. We also show that the scheduling windows for jobs $J_1^1$ and $J_1^2$, and job $J_4$ are continuously busy (Lemmas 13 and 14, respectively).

**Lemma 12** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then the interval $[A_5, A_5 + 6)$ is a continuously busy interval on processor $\pi_1$ in schedule $S'_I$ for jobs of tasks $\tau^A \cup \{\tau_5\}$. More formally,*

$$\sum_{\tau_j \in \tau^A \cup \{\tau_5\}} W_j(S'_I, \pi_1, A_5, A_5 + 6) = 6 \tag{31}$$

**Proof:** Again, by Lemma 5, $J_5$ executes on $\pi_2$ for $\alpha > 0$ time in $S'_I$. Since $S_I$ moves as much execution of $J_5$ from $\pi_2$ to $\pi_1$, this implies for all time $t \in [A_5, A_5 + 6)$ at which $\pi_1$ is not executing $J_5$, it must be executing jobs of $\tau^A$. ■

**Lemma 13** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then the intervals $[A_1^1, A_1^1 + 2)$ and $[A_1^2, A_1^2 + 2)$ are a continuously busy intervals on processor $\pi_1$ in schedule $S'_I$ for jobs of tasks $\tau^A \cup \{\tau_5\}$. More formally, for $k \in \{1, 2\}$,*

$$\sum_{\tau_j \in \tau^A \cup \{\tau_5\}} W_j(S'_I, \pi_1, A_1^k, A_1^k + 2) = 2 \tag{32}$$

**Proof:** Any job of $\tau_1$ in $I$ must execute continuously from its arrival to deadline because $e_1 = d_1 = 2$. Thus, since $\tau_1$ is scheduled on processor $\pi_1$ in $S'_I$ and since $S'_I$ is valid, $J_1^1$ executes continuously on $\pi_1$ over $[A_1^1, A_1^1 + 2)$ and $J_1^2$ executes continuously on $\pi_1$ over $[A_1^2, A_1^2 + 2)$. ■

**Lemma 14** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then the interval $[A_4, A_4 + 4)$ is a continuously busy interval on processor $\pi_1$ in schedule $S'_I$ for jobs of tasks $\tau^A \cup \{\tau_5\}$. More formally,*

$$\sum_{\tau_j \in \tau^A \cup \{\tau_5\}} W_j(S'_I, \pi_1, A_4, A_4 + 4) = 4 \tag{33}$$

**Proof:** Lemma 11 implies that both $J_1^1$'s and $J_1^2$'s scheduling window intersects with the interval $[A_5, A_5 + 6)$. Since $J_1^1$'s arrival precedes $J_1^2$'s arrival, the lemma also implies that $[A_1^1 + 2, A_1^2) \subset [A_5, A_5 + 6)$; in words, the time interval between the deadline of $J_1^1$ to the arrival of $J_1^2$ is a proper subset of the $J_5$'s scheduling window. The interval $[A_1^1 + 2, A_1^2)$ is between the scheduling window of two consecutive jobs of $\tau_1$; therefore, no job of $\tau_1$ can execute in $S'_I$ during the interval $[A_1^1+2, A_1^2)$. Due to the period and relative deadline parameter of $\tau_1$, the length of this interval must be at least three time units (i.e., $A_1^2 - A_1^1 - 2 \geq 3$). By Lemma 12 and $[A_1^1+2, A_1^2) \subset [A_5, A_5+6)$, the interval $[A_1^1+2, A_1^2)$ is continuously busy executing jobs of $\tau_4$ and $\tau_5$ on processor $\pi_1$ in schedule $S'_I$. Lemma 5 implies that $J_5$ can execute on processor $\pi_1$ for at most $2 - \alpha$ time in schedule $S'_I$. Thus, $J_4$ must execute for at least $1 + \alpha$ time units on processor $\pi_1$ over the interval $[A_1^1 + 2, A_1^2)$ in schedule $S'_I$; i.e.,

$$W_4(S'_I, \pi_1, A_1^1 + 2, A_2^1) \geq 1 + \alpha \tag{34}$$

Lemma 10 implies that $[A_4, A_4 + 4) \cap [A_5, A_5 + 6) \neq \emptyset$. The above equation (Equation 34) and the validity of schedule $S'_I$ implies that $[A_4, A_4 + 4) \cap [A_1^1 + 2, A_1^2) \neq \emptyset$. From these statements, we can reason about the work of $\tau^A \cap \{\tau_5\}$ over $J_4$'s scheduling window. There are two separate main cases we consider: 1) if $J_4$'s scheduling window is completely contained within $J_5$'s scheduling window; 2) $J_4$'s scheduling window is not completely contained in $J_5$'s scheduling window. We will show that each of the cases imply Equation 33. The case analysis is below.

1. $[A_4, A_4 + 4) \subseteq [A_5, A_5 + 6)$: Lemma 12 states that $\pi_1$ is continuously busy executing jobs of $\tau^A \cap \{\tau_5\}$ over $[A_5, A_5 + 6)$. Thus, Equation 33 follows trivially.

2. $[A_4, A_4 + 4) \not\subseteq [A_5, A_5 + 6)$: Given this case, there are two possibilities. Either the job of $\tau_4$ is released before $A_5$ or it is released after $A_5$. More formally, the subcases are:

   a) $A_4 < A_5 < A_4 + 4$: In this case, Equation 34 and $[A_4, A_4 + 4) \cap [A_1^1 + 2, A_1^2) \neq \emptyset$ imply that $J_4$'s deadline must be at least $1 + \alpha$ after $A_5$ (i.e., $A_4 + 4 \geq A_5 + 1 + \alpha$). Since $[A_1^1, A_1^1 + 2) \cap [A_5, A_5 + 6) \neq \emptyset$ (Lemma 11), it must also be that $[A_4, A_4 + 4) \cap [A_1^1, A_1^1 + 2) \neq \emptyset$. Otherwise, if $[A_4, A_4 + 4) \cap [A_1^1, A_1^1 + 2) = \emptyset$, then $J_1^1$ must arrive after $J_4$'s deadline, in order to still overlap with $J_5$'s scheduling window. In this case, the earliest $J_4$'s deadline may occur is $1 + \alpha$ units after $A_5$; hence, $A_1^1 \geq A_5 + 1 + \alpha$. However, this inequality and the minimum separation between $J_1^1$ and $J_1^2$ imply $A_1^2 \geq A_1^1 + 5 \geq A_5 + 6 + \alpha$. This further implies $[A_1^2, A_1^2 + 2) \cap [A_5, A_5 + 6) = \emptyset$ which contradicts Lemma 11. So given that $[A_4, A_4 + 4) \cap [A_1^1, A_1^1 + 2) \neq \emptyset$ is true, we may consider three additional subcases regarding the execution of $J_4$ in relation to $J_1^1$'s absolute deadline.

      i) $J_4$ *executes entirely after* $A_1^1 + 2$ *(i.e., $J_4$ executes only in the interval* $[A_1^1 + 2, A_4 + 4)$*)*: Because the execution requirement of $J_4$ is two and $[A_4, A_4 + 4) \cap [A_1^1 + 2, A_1^2) \neq \emptyset$, it must be that $A_4 \in [A_1^1, A_1^1 + 2)$; otherwise, there length of the interval $[A_1^1, A_4 + 4)$ would leave insufficient time for $J_4$ to execute. Since $A_4 < A_5$ in this case, $A_1^1 \leq A_4 < A_5$. Thus, the interval $[A_4, A_4 + 4)$ is a subset of $[A_1^1, A_5 + 6)$. By Lemma 13, $\pi_1$ is continuously busy executing $J_1^1$ during $[A_1^1, A_1^1 + 2)$. By Lemma 12, $\pi_1$ is continuously busy executing jobs of $\tau^A \cap \{\tau_5\}$ during $[A_5, A_5 + 6)$. It must be that $\pi_1$ is also continuously busy executing jobs of $\tau^A \cap \{\tau_5\}$ over the interval $[A_4, A_4 + 4)$ in schedule $S'_I$, because it is a subset of the union of these two continuously busy intervals. This implies Equation 33.

      ii) $J_4$ *executes both before* $A_1^1$ *and after* $A_1^1 + 2$: Observe that job $J_1^1$ executes continuously over $[A_1^1, A_1^1 + 2)$. Since $J_4$ executes both before and after $A_1^1$ and $S'_I$ is valid, it follows that $[A_1^1, A_1^1 + 2) \subset [A_4, A_4 + 4)$. Thus, $J_4$ must continuously execute on processor $\pi_1$ over the intervals $[A_4, A_1^1)$ and $[A_1^1 + 2, A_4 + 4)$ in schedule $S'_I$ to complete by its deadline. Since processor $\pi_1$ is continuously busy executing either $J_4$ or $J_1^1$ over the intervals $[A_4, A_1^1)$, $[A_1^1, A_1^1 + 2)$ and $[A_1^1 + 2, A_4 + 4)$, it

is continuously busy over the interval $[A_4, A_4 + 4)$ in schedule $S'_I$ executing jobs of $\tau_A \cap \{\tau_5\}$. This implies Equation 33.

iii) $J_4$ *executes entirely before* $A_1^1$: Equation 34 implies that this case is impossible.

b) $A_5 + 2 < A_4 < A_5 + 6$: Symmetric to Case a.

■

We now concentrate on identifying the longest continuously busy interval on processor $\pi_1$ for tasks $\tau^A \cap \{\tau_5\}$ that contains the interval $[A_5, A_5 + 6)$. By identifying this interval, we may more easily reason about the amount of execution of jobs of $\tau_2$ or $\tau_3$ on processor $\pi_1$ in schedule $S'_I$. We begin by defining $t_{start}$ which we will show is the start of the longest continuously busy interval containing $[A_5, A_5 + 6)$.

$$t_{start} \stackrel{\text{def}}{=} \min\{A_1^1, A_4, A_5\} \tag{35}$$

The next lemma shows that the the interval $[t_{start}, t_{start}+8-\alpha)$ is continuously busy on $\pi_1$ for tasks $\tau^A \cup \{\tau_5\}$; Lemma 16 will show that $[t_{start}, t_{start} + 8 - \alpha)$ is, in fact, the maximum continuously busy interval that contains $[A_5, A_5+6)$ because the time instants immediately before $t_{start}$ or immediately after $t_{start}+8-\alpha$ cannot execute jobs of $\tau^A \cup \{\tau_5\}$.

**Lemma 15** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then the interval $[t_{start}, t_{start} + 8 - \alpha) \supset [A_5, A_5 + 6)$ is a continuously busy interval on processor $\pi_1$ with respect to schedule $S'_I$ and tasks $\tau^A \cup \{\tau_5\}$. More formally,*

$$\sum_{\tau_j \in \tau^A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_{start}, t_{start} + 8 - \alpha) = 8 - \alpha, \tag{36}$$

*Furthermore, jobs $J_1^1$, $J_1^2$, $J_4$, and $J_5$ are the only jobs to execute on processor $\pi_1$ over $[t_{start}, t_{start} + 8 - \alpha)$ in schedule $S'_I$.*

**Proof:** Lemmas 10 and 11 imply that the scheduling windows of jobs $J_4$, $J_1^1$, and $J_1^2$ intersect with the scheduling window of job $J_5$. Lemmas 12, 13, and 14 imply that the scheduling windows of jobs $J_5$, $J_1^1$, $J_1^2$, and $J_4$ are continuously busy intervals on processor $\pi_1$ in schedule $S'_I$ for tasks $\tau^A \cap \{\tau_5\}$. Thus, the union of the scheduling windows of $J_5$, $J_1^1$, $J_1^2$, and $J_4$ is also a continuously busy interval on $\pi_1$ for $\tau^A \cap \{\tau_5\}$; i.e., $[A_5, A_5+6) \cup [A_1^1, A_1^1+2) \cup [A_1^2, A_1^2+2) \cup [A_4, A_4+4) = \big[\min\{A_5, A_1^1, A_4\}, \max\{A_5+6, A_1^2+2, A_4+4\}\big)$ is a continuously busy interval on $\pi_1$.

We will now show that $\big[\min\{A_5, A_1^1, A_4\}, \max\{A_5+6, A_1^2+2, A_4+4\}\big)$ equals the interval $[t_{start}, t_{start} + 8-\alpha)$. Obviously, by definition of Equation 35, $\min\{A_5, A_1^1, A_4\}$ equals $t_{start}$; so, we must show that $\max\{A_5 + 6, A_1^2 + 2, A_4 + 4\}$ equals $t_{start} + 8 - \alpha$. Lemma 12 shows that processor $\pi_1$ over the interval $[A_5, A_5 + 6)$ in $S'_I$ executes only jobs $J_1^1$, $J_1^2$, $J_4$, and $J_5$. The busy interval $\big[\min\{A_5, A_1^1, A_4\}, \max\{A_5+6, A_1^2+2, A_4+4\}\big)$ must include these jobs, whose total execution on processor $\pi_1$ in schedule $S'_I$ equals $2 + 2 + 2 + 2 - \alpha = 8 - \alpha$. Because the execution of these jobs must complete in the busy interval, $\max\{A_5 + 6, A_1^2 + 2, A_4 + 4\}$ must be at least $t_{start} + 8 - \alpha$. If $\max\{A_5 + 6, A_1^2 + 2, A_4 + 4\}$ exceeds $t_{start} + 8 - \alpha$, then some job $\tau_A \cup \{\tau_5\}$ (other than $J_1^1$, $J_1^2$, $J_4$ or $J_5$) must have a scheduling window that overlaps $[t_{start}, t_{start} + 8 - \alpha)$; otherwise, the interval $\big[\min\{A_5, A_1^1, A_4\}, \max\{A_5+6, A_1^2+2, A_4+4\}\big)$ is not continuously busy for tasks $\tau^A \cap \{\tau_5\}$. However, Observation 1 implies that such a job cannot exist. Therefore, $\max\{A_5 + 6, A_1^2 + 2, A_4 + 4\}$ equals $t_{start} + 8 - \alpha$, implying Equation 36. ■

Since the busy interval $[t_{start}, t_{start}+8-\alpha)$ includes the entire execution from jobs $J_1^1$, $J_1^2$, $J_4$, and $2-\alpha$ units of execution from $J_5$, there is an idle period (with respect to tasks $\tau^A \cup \{\tau_5\}$) before and after $[t_{start}, t_{start}+8-\alpha)$. The following lemma exactly characterizes these idle periods.

**Lemma 16** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then no job of $\tau^A \cap \{\tau_5\}$ executes on processor $\pi_1$ in either the interval $[t_{start} - 2 - \alpha, t_{start})$ or $[t_{start} + 8 - \alpha, t_{start} + 10)$. More formally,*

$$\sum_{\tau_j \in \tau^A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_{start} - 2 - \alpha, t_{start}) = 0, \tag{37}$$

*and*

$$\sum_{\tau_j \in \tau^A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_{start} + 8 - \alpha, t_{start} + 10) = 0. \tag{38}$$

**Proof:** We begin with Equation 37: we show that $S'_I$ does not execute any jobs of $\tau^A \cap \{\tau_5\}$ on processor $\pi_1$ during the interval $[t_{start} - 2, t_{start})$. Equation 38 can be shown by a symmetric argument. Observe that

$$W_1(S'_I, \pi_1, A_1^1 - 3, A_1^1) = 0 \tag{39}$$

because the period of $\tau_1$, $p_1$ equals five and the relative deadline, $d_1$, equals two. Recall from the proof of Lemma 14 that the interval $[A_1^1 + 2, A_1^2)$ is a subset of $[A_5, A_5 + 6)$ and that $A_1^2 - A_1^1 - 2 \geq 3$. So, the interval $[A_1^1 + 2, A_1^2)$ is continuously busy on processor $\pi_1$ executing either $J_4$ or $J_5$:

$$\sum_{\tau_j \in \{\tau_4, \tau_5\}} W_j(S'_I, \pi_1, A_1^1 + 2, A_2^1) \geq 3.$$

Since at least three units of $J_4$ and $J_5$ must execute in the interval $[A_1^1 + 2, A_2^1)$ and the total execution of $J_4$ and $J_5$ on $\pi_1$ is $4 - \alpha$, this leaves at most $1 - \alpha$ units left to execute either before $A_1^1$ and/or after $A_2^1 + 2$. This implies

$$t_{start} \geq A_1^1 - 1 + \alpha. \tag{40}$$

Equations 39 and 40 imply that the latest another job of $\tau_1$ (that precedes $J_1^1$) could execute prior to $t_{start}$ is $t_{start} - 2 - \alpha$. Since $\tau_5$ and $\tau_4$ have periods equal to 100, and they release jobs contained within $[t_{start}, t_{start} + 8 - \alpha)$, they are not current in the interval $[t_{start} - 2 - \alpha, t_{start})$. Therefore,

$$\sum_{\tau_j \in \tau^A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_{start} - 2 - \alpha, t_{start}) = 0.$$

■

Processor $\pi_1$ can also be shown to be busy during the scheduling windows for jobs of $\tau^B$. Lemma 17 shows that there is no idle time on $\pi_1$ in $S'_I$ over the scheduling window for any job of $\tau^B$ that executes a non-zero amount of time on $\pi_2$. Lemma 18 will show for any interval on $\pi_1$ that is continuously busy for jobs of $\tau_2$ and $\tau_3$, no job of these two tasks can execute on processor $\pi_1$ in the same interval.

**Lemma 17** *For any $t \geq 0$ where $S'_I(\pi_2, t) \neq \perp$, let $J_k = S'_I(\pi_2, t)$ where $J_k = (A_k, E_k, D_k) \in I$. For all $t' \in [A_k, A_k + D_k)$,*

$$S'_I(\pi_1, t') \neq \perp. \tag{41}$$

**Proof:** Note that $J_k \in I$ must have been generated by a task of $\tau^B$ in order to be executed on $\pi_2$ in $S'_I$. By construction of $S'_I$, as much of the execution of $J_k$ has been moved from $\pi_2$ to $\pi_1$ (with respect to the idle times on processor $\pi_1$ in schedule $S_I$). Since $J_k$ executed on $\pi_2$ for a non-zero amount of time there is no further unused idle time in $[A_k, A_k + D_k)$; thus, $S'_I(\pi_1, t') \neq \perp$ for all $t' \in [A_k, A_k + D_k)$ which implies the lemma. ■

**Lemma 18** *For any interval $[t_1, t_2)$ where $0 \leq t_1 < t_2$ where processor $\pi_1$ is continuously busy in $S'_I$ with respect to jobs of $\tau_2$ and $\tau_3$, then no job of $\tau_2$ or $\tau_3$ is executed on processor $\pi_2$ in $S'_I$ over $[t_1, t_2)$. More formally,*

$$\sum_{\tau_j \in \{\tau_2, \tau_3\}} W_j(S'_I, \pi_2, t_1, t_2) = 0 \tag{42}$$

**Proof:** Assume that $[t_1, t_2)$ is a continuously busy interval on processor $\pi_1$ for $\tau_2$ and $\tau_3$. Thus, for each $t \in [t_1, t_2)$, either $S'_I(\pi_1, t, \tau_2) = 1$ or $S'_I(\pi_1, t, \tau_3) = 1$. We will show in either case that $S'_I(\pi_2, t) = \perp$. If $S'_I(\pi_1, t, \tau_2) = 1$, then $S_I(\pi_2, t, \tau_2) = 1$ in the original schedule, since $e_1 = d_1 = 1$. Because $S'_I$ is a valid schedule $S'_I(\pi_2, t, \tau_2) = 0$. Further $S'_I(\pi_2, t, \tau_3) = 0$ due to the fact that $S'_I$ schedules jobs of $\tau^B$ on processor $\pi_2$ only at times that they were scheduled on processor $\pi_2$ in the original schedule $S_I$. Thus, $S'_I(\pi_1, t, \tau_2) = 1$ implies that $S'_I(\pi_2, t) = \perp$.

If $S'_I(\pi_1, t, \tau_3) = 1$, then $S'_I(\pi_2, t, \tau_3) = 0$ due to the validity of $S'_I$. Since we move as much execution of $\tau_2$ from $\pi_2$ to $\pi_1$ before moving $\tau_3$'s execution, a job of task $\tau_3$ cannot be executing on processor $\pi_1$ at the same time that $\tau_2$ is executing on processor $\pi_2$; otherwise, since $\tau_2$ is only scheduled at points during which $\pi_1$ is idle in the original schedule $S_I$, we could have moved more execution of $\tau_2$ to processor $\pi_1$. Thus, we have shown that $S'_I(\pi_2, t, \tau_2) = 0$ for this case, implying $S'_I(\pi_2, t) = \perp$ and the lemma. ■

For the final lemma of Step 3 (Lemma 22), we derive constraints on the arrival times of $J_5$ and $J_6$. In fact, if $J_6$ cannot complete in schedule $S'_I$, then $J_5$'s scheduling window cannot be contained within $J_6$'s scheduling window. Furthermore, we show that either $J_5$ arrives at least two time units before the arrival of $J_6$, or that $J_5$ has a deadline at least two units after $J_6$'s deadline. Before we can prove Lemma 22, we require three technical lemmas: Lemmas 19 and 20 are concerned with the execution of jobs $J_3^1$ and $J_3^2$ in relation to the intervals $[t_{start}, t_{start}+8-\alpha)$ and $[A_6, A_6+8)$; Lemma 21 describes the relative overlap of the intervals $[t_{start}, t_{start}+8-\alpha)$ and $[A_6, A_6+8)$.

**Lemma 19** *Given that $S'_I$ does not have sufficient idle time over $[A_6, A_6+8)$ to completely execute $J_6$. If jobs $J_3^1$ and $J_3^2$ both have their scheduling window intersect with $[t_{start}, t_{start}+8-\alpha)$ and $0 < t_{start} - A_3^1 < \alpha$, then*

$$[t_{start}+8-\alpha, A_3^1+8) \subset [A_3^2, A_3^2+2) \tag{43}$$

**Proof:** Observe that since $t_{start} - A_3^1 < \alpha$, the inequality

$$t_{start}+8-\alpha < A_3^1+8 \tag{44}$$

must hold. Since $J_3^2$'s scheduling window intersects with $[t_{start}, t_{start}+8-\alpha)$, the following inequality must be true:

$$A_3^2 < t_{start}+8-\alpha. \tag{45}$$

The period parameter of $\tau_3$ ($p_3 = 6$) implies $A_3^1 + 6 \leq A_3^2$. This inequality along with $t_{start} - \alpha < A_3^1$ implies

$$\begin{aligned} &t_{start}+6-\alpha < A_3^2 \\ &\quad \Rightarrow t_{start}+8-\alpha < A_3^2+2. \end{aligned} \tag{46}$$

Furthermore, $A_3^1 + 6 \leq A_3^2$ implies

$$A_3^1+8 \leq A_3^2+2 \tag{47}$$

Inequalities 44, 45, 46, and 47 taken together imply Equation 43 of the lemma. ■

**Lemma 20** *Given that $S'_I$ does not have sufficient idle time over $[A_6, A_6+8)$ to completely execute $J_6$. If jobs $J_3^1$ and $J_3^2$ both have their scheduling window intersect with $[t_{start}, t_{start}+8-\alpha)$, then there exists $t' \geq 0$ such that*

$$[t', t'+10) \supset [A_6, A_6+8), \tag{48}$$

*and*

$$W_2(S'_I, \pi_1, t', t'+10) + W_3(S'_I, \pi_1, t', t'+10) \geq \alpha. \tag{49}$$

**Proof:** Because both $J_3^1$ and $J_3^2$ have scheduling windows that overlap with $[t_{start}, t_{start}+8-\alpha)$, the interval between the scheduling windows of $J_3^1$ and $J_3^2$ must be completely contained in $[t_{start}, t_{start}+8-\alpha)$ (i.e., $[A_3^1+2, A_3^2) \subset [t_{start}, t_{start}+8-\alpha)$). The period and relative deadline parameter of $\tau_3$ ($p_3 = 6$ and $d_3 = 2$) imply that $A_3^2-(A_3^1+2) \geq 4$. Therefore, total intersection between the $[t_{start}, t_{start}+8-\alpha)$ and the scheduling windows of $J_3^1$ and $J_3^2$ is at most $4-\alpha$. Since the aggregate length of the scheduling windows for $J_3^1$ and $J_3^2$ is four, the total remaining portion of $J_3^1$ and $J_3^2$'s scheduling windows that do not overlap with $[t_{start}, t_{start}+8-\alpha)$ is at least $\alpha$. This remaining portion of the scheduling windows of $J_3^1$ and $J_3^2$ must overlap with either $[t_{start}-2-\alpha, t_{start})$ or $[t_{start}+8-\alpha, t_{start}+10)$ which, by Lemma 16, does not contain the execution of jobs of $\tau^A \cup \{\tau_5\}$.

According to Lemma 7, $J_3^1$ and $J_3^2$ execute on $\pi_2$ over $[A_6, A_6+8)$ for at least $2-\alpha$ time units. Since the execution requirement of each job of $\tau_3$ is one (i.e., $e_3 = 1$), both $J_3^1$ and $J_3^2$ must each execute on $\pi_2$ over $[A_6, A_6+8)$ for at least $1-\alpha$ time units. Thus, the scheduling window of both $J_3^1$ and $J_3^2$ must each overlap with $[A_6, A_6+8)$ for at least $1-\alpha$ time units. Therefore, the earliest that $J_3^1$ could arrive is at time $A_6-1-\alpha$ (otherwise, $J_3^1$ would overlap with $[A_6, A_6+8)$ less than $1-\alpha$ time units). Similarly, the latest that $J_3^2$ could have its deadline is $A_6+9+\alpha$. More formally,

$$A_3^1 \geq A_6 - 1 - \alpha, \tag{50}$$

and

$$A_3^2 + 2 \leq A_6 + 9 + \alpha. \tag{51}$$

We now consider three cases based on how $J_3^1$ and $J_3^2$ intersect with $[t_{start}, t_{start}+8-\alpha)$. In each case, we will prove that there exists a $t \geq 0$ that satisfies the conditions of Equations 48 and 49. The three cases are:

**Case I)** $J_3^1$'s scheduling window is completely contained within $[t_{start}, t_{start}+8-\alpha)$;

**Case II)** $J_3^2$'s scheduling window is completely contained within $[t_{start}, t_{start}+8-\alpha)$; or

**Case III)** Neither $J_3^1$'s nor $J_3^2$'s scheduling window is completely contained within $[t_{start}, t_{start}+8-\alpha)$.

(Observe that the argument of the first paragraph of the proof implies that both $J_3^1$ and $J_3^2$ cannot have their scheduling windows completely contained within $[t_{start}, t_{start}+8-\alpha)$).

**Analysis for Case I.** Both $[A_3^1, A_3^1+2)$ and $[A_3^1+2, A_3^2)$ are proper subsets of $[t_{start}, t_{start}+8-\alpha)$. Thus, by the argument of the first paragraph, at least $\alpha$ of $J_3^2$'s scheduling window must intersect with $[t_{start}+8-\alpha, t_{start}+10)$. More precisely, $[t_{start}+8-\alpha, t_{start}+8) \subset [A_3^2, A_3^2+2)$. By Lemma 16, jobs of $\tau^A \cup \{\tau_5\}$ do not execute during $[t_{start}+8-\alpha, t_{start}+8)$. Lemma 7 implies that $J_3^2$ must execute on processor $\pi_2$ for some non-zero amount of time in schedule $S'_I$. According to Lemma 13 and the fact that $\tau^A \cup \{\tau_5\}$ cannot execute during this interval, $[t_{start}+8-\alpha, t_{start}+8)$ must be continuously busy on processor $\pi_1$ with respect to jobs of $\tau_2$ and $\tau_3$. By Equation 50 (i.e., $A_3^1 \geq A_6 - 1 - \alpha$) and the period parameter of $\tau_3$ (i.e., $p_3 = 6$), $A_3^2 \geq A_6 + 5 - \alpha$ must be true; since $\alpha \leq 1$ (by Lemma 6),

$$A_6 \leq A_3^2. \tag{52}$$

Equation 51 states that $A_3^2 + 2 \leq A_6 + 9 + \alpha$; since $\alpha \leq 1$, it must be that

$$A_3^2 + 2 \leq A_6 + 10. \tag{53}$$

Equations 52 and 53 together imply $[A_6, A_6{+}10) \supset [A_3^2, A_3^2{+}2)$. Furthermore, we have shown that $[A_3^2, A_3^2{+}2) \supset [t_{start} + 8 - \alpha, t_{start} + 8)$; thus, $[A_6, A_6 + 10) \supset [t_{start} + 8 - \alpha, t_{start} + 8)$. Since $[t_{start} + 8 - \alpha, t_{start} + 8)$ is continuously busy on $\pi_1$ for $\alpha$ time units executing jobs of $\tau_2$ and $\tau_3$ and $[A_6, A_6 + 8) \subset [A_6, A_6 + 10)$, the interval $[A_6, A_6 + 10)$ satisfies both Equations 48 and 49 of the lemma.

**Analysis for Case II.** This case is exactly symmetric to Case II.

**Analysis for Case III.** Consider the interval $[A_3^1, A_3^1 + 8)$. We first show that $[A_3^1, A_3^1 + 8)$ contains at least $\alpha$ units of execution for jobs of $\tau_2$ and $\tau_3$ on processor $\pi_1$. Since $J_3^1$ intersects $[t_{start}, t_{start} + 8 - \alpha)$ and $d_3 = 2$, it must be that $t_{start} - 2 < A_3^1 < t_{start}$, which implies that $t_{start} - A_3^1 < 2$. Thus, the interval $[A_3^1, t_{start})$ is contained within $[t_{start} - 2 - \alpha, t_{start})$ which by Lemma 16 cannot contain the execution of jobs of $\tau^A \cup \{\tau_5\}$ on processor $\pi_1$. According to Lemma 7, $J_3^1$ must execute on processor $\pi_2$ for some non-zero amount of time which implies that there must exist a time $t \in [A_3^1, A_3^1 + 2)$ such that $S'_I(\pi_2, t) = J_3^1 (\neq \perp)$. Note the preceding statement satisfies the supposition of Lemma 17; so, for all $t' \in [A_3^1, A_3^1 + 2)$, $S'_I(\pi_1, t') \neq \perp$. By Lemma 17 and the fact that jobs of $\tau^A \cup \{\tau_5\}$ do not execute on processor $\pi$ over $[A_3^1, t_{start})$, processor $\pi_1$ must be continuously busy over the interval $[A_3^1, t_{start})$ executing only jobs of $\tau_2$ and $\tau_3$. If the interval length of $[A_3^1, t_{start})$ is greater or equal to $\alpha$, then we have shown that $[A_3^1, A_3^1 + 8)$ contains at least $\alpha$ units of execution of $\tau_2$ and $\tau_3$ on processor $\pi_1$. If the interval length of $[A_3^1, t_{start})$ is less than $\alpha$, then $[A_3^1, A_3^1 + 8) \supset [t_{start}, t_{start} + 8 - \alpha)$. Additionally, Lemma 19 implies that interval $[t_{start} + 8 - \alpha, A_3^1 + 8)$ must be contained within $[A_3^2, A_3^2 + 2)$. $J_3^2$ must execute on processor $\pi_2$ for some non-zero time by Lemma 7. Lemma 17 implies then that $\pi_1$ is continuously busy over $[A_3^2, A_3^2 + 2)$. However, $[t_{start} + 8 - \alpha, A_3^1 + 8) \subset [t_{start} + 8 - \alpha, A_3^2 + 2)$ cannot contain the execution of jobs of $\tau^A \cup \{\tau_5\}$ (by Lemma 16). Thus, the interval $[t_{start} + 8 - \alpha, A_3^1 + 8)$ is continuously busy executing only jobs of $\tau_2$ and $\tau_3$ on processor $\pi_1$ in schedule $S'_I$. Therefore, $[A_3^1, A_3^1 + 8)$ contains intervals (namely $[A_3^1, t_{start})$ and/or $[t_{start} + 8 - \alpha, A_3^1 + 8)$) of total length $\alpha$ that are continuously busy executing jobs of $\tau_2$ and $\tau_3$ on processor $\pi_1$.

Continuing our analysis of Case III, we will now show that the interval $\big[\min(A_3^1, A_6), \min(A_3^1, A_6) + 10\big)$ is a superset for both intervals $[A_6, A_6 + 8)$ and $[A_3^1, A_3^1 + 8)$, and thus satisfies Equations 48 and 49 of the lemma. There are two subcases to consider:

*Subcase III.A)* $A_3^1 \leq A_6$; or

*Subcase III.B)* $A_3^1 > A_6$.

For Subcase III.A, $\big[\min(A_3^1, A_6), \min(A_3^1, A_6) + 10\big)$ is equivalent to the interval $[A_3^1, A_3^1 + 10)$. Equation 50 states that $A_3^1 \geq A_6 - 1 - \alpha$. This implies that $A_3^1 + 10 \geq A_6 + 9 - \alpha$. Since $\alpha > 0$ (Lemma 5), $[A_6, A_6 + 8) \subset [A_3^1, A_3^1 + 10)$. Furthermore, $[A_3^1, A_3^1 + 8)$ is obviously a subset of $[A_3^1, A_3^1 + 10)$.

For Subcase III.B, $\big[\min(A_3^1, A_6), \min(A_3^1, A_6) + 10\big)$ is equivalent to the interval $[A_6, A_6 + 10)$. Equation 51 states that $A_3^2 + 2 \leq A_6 + 9 + \alpha \Rightarrow A_3^2 \leq A_6 + 7 + \alpha$. Due to the period parameter for $\tau_3$ (i.e., $p_3 = 6$), $A_3^1 \leq A_6 + 1 + \alpha$. Adding eight to both sides of the inequality implies, $A_3^1 + 8 \leq A_6 + 9 + \alpha$. Since $\alpha \leq 1$ (Lemma 6), $[A_3^1, A_3^1 + 8) \subset [A_6, A_6 + 10)$. Furthermore, $[A_6, A_6 + 8)$ is obviously a subset of $[A_6, A_6 + 10)$. In both the subcases, we have shown that both $[A_3^1, A_3^1{+}8)$ and $[A_6, A_6{+}8)$ are subsets of $\big[\min(A_3^1, A_6), \min(A_3^1, A_6) + 10\big)$. Thus, $\big[\min(A_3^1, A_6), \min(A_3^1, A_6) + 10\big)$ satisfies the conditions of Equations 48 and 49 of the lemma. ■

**Lemma 21** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then*

$$(A_6 - 2 > t_{start}) \vee (t_{start} > A_6 + 2 + \alpha) \tag{54}$$

**Proof:** We will prove the lemma by contradiction; that is, we will show that if

$$A_6 - 2 \leq t_{start} \leq A_6 + 2 + \alpha \tag{55}$$

is true, then we reach a logical contradiction.

By Lemma 16, the intervals $[t_{start} - 2 - \alpha, t_{start})$ and $[t_{start} + 8 - \alpha, t_{start} + 10)$ do not contain the execution of jobs of task $\tau^A \cup \{\tau_5\}$ on processor $\pi_1$ in schedule $S'_I$. Equation 55 states that $t_{start} \leq A_6 + 2 + \alpha$. This implies that $A_6 \geq t_{start} - 2 - \alpha$. Therefore, $[A_6, t_{start})$ is a subset of $[t_{start} - 2 - \alpha, t_{start})$ and hence no jobs of $\tau^A \cup \{\tau_5\}$ may execute in $[A_6, t_{start})$. (Please note that $[A_6, t_{start})$ may be empty if $t_{start} \leq A_6$). Similarly, since $t_{start} \geq A_6 - 2$, then $t_{start} + 10 \geq A_6 + 8$; this implies that interval $[t_{start} + 8 - \alpha, A_6 + 8)$ also does not contain the execution of jobs of $\tau^A \cup \{\tau_5\}$. (Again, $[t_{start} + 8 - \alpha, A_6 + 8)$ may be empty if $t_{start} + 8 - \alpha \geq A_6 + 8$). Thus, the only times during which processor $\pi_1$ executes jobs of $\tau^A \cup \{\tau_5\}$ over $[A_6, A_6 + 8)$ in schedule $S'_I$ is over the subinterval $[t_{start}, t_{start} + 8 - \alpha) \cap [A_6, A_6 + 8)$.

Lemma 7 implies that two jobs of $\tau_3$, namely $J_3^1, J_3^2 \in I$, must execute on processor $\pi_2$ in schedule $S'_I$ over the interval $[A_6, A_6 + 8)$ for strictly more than $2 - \alpha$ time units. We now consider three possible subcases regarding the intersection between the interval $[t_{start}, t_{start} + 8 - \alpha)$ and the scheduling windows of $J_3^1$ and $J_3^2$.

**Case I)** Both the scheduling windows of $J_3^1$ and $J_3^2$ intersect with $[t_{start}, t_{start} + 8 - \alpha)$;

**Case II)** only one of either $J_3^1$ or $J_3^2$ has a scheduling window that intersects with $[t_{start}, t_{start} + 8 - \alpha)$; or

**Case III)** neither $J_3^1$ nor $J_3^2$ intersect with $[t_{start}, t_{start} + 8 - \alpha)$.

For Case I, Lemma 20 implies that there exists an interval $[t', t' + 10)$ such that $W_2(S'_I, \pi_1, t', t' + 10) + W_3(S'_I, \pi_1, t', t' + 10) \geq \alpha$ and $[t', t' + 10) \supset [A_6, A_6 + 8)$. Observation 1 states that the most that jobs of $\tau_2$ and $\tau_3$ can execute in $S'_I$ over $[t', t' + 10)$ is four units. Since $\tau_2$ and $\tau_3$ execute for at least $\alpha$ time units on processor $\pi_1$ over $[t', t' + 10)$, $\tau_2$ and $\tau_3$ can execute for at most $4 - \alpha$ time units on processor $\pi_2$ over the same interval. Because $[A_6, A_6 + 8) \subset [t', t' + 10)$, the preceding statement implies that $\tau_2$ and $\tau_3$ execute for at most $4 - \alpha$ time units on $\pi_2$ over $[A_6, A_6 + 8)$ in $S'_I$. However, this directly contradicts Lemma 9.

For Case II, without loss of generality, assume that $J_3^1$ is the job that does not intersect with $[t_{start}, t_{start} + 8 - \alpha)$. Since $J_3^1$ does not intersect with $[t_{start}, t_{start} + 8 - \alpha)$, the interval $[A_6, A_6 + 8) \cap [A_3^1, A_3^1 + 2)$ does not contain the execution of jobs of $\tau^A \cup \{\tau_5\}$ on processor $\pi_1$ in schedule $S'_I$ (according to the argument at the beginning of Case I about the execution of jobs of $\tau^A \cup \{\tau_5\}$ over $[A_6, A_6 + 8)$). Lemma 7 implies that $J_3^1$ executes on $\pi_2$ over $[A_6, A_6 + 8)$. Lemma 17 thus, implies that $\pi_1$ is continuously busy over $[A_3^1, A_3^1 + 2)$. However, we have just argued that $\tau^A \cup \{\tau_5\}$ do not execute on $\pi_1$ over $[A_6, A_6 + 8) \cap [A_3^1, A_3^1 + 2)$. Thus, $\pi_1$ is continuously busy over $[A_6, A_6 + 8) \cap [A_3^1, A_3^1 + 2)$ for $\tau_2$ and $\tau_3$. Lemma 18 implies that $\pi_2$ is idle over the interval $[A_6, A_6 + 8) \cap [A_3^1, A_3^1 + 2)$ for tasks $\tau_2$ and $\tau_3$. However, this contradicts the earlier statement that $J_3^1$ must have executed on $\pi_2$ over $[A_6, A_6 + 8)$. Thus, this case is not possible, since we have reached a contradiction to Lemma 7

The proof of Case III is identical to Case II, except neither $J_3^1$ nor $J_3^2$ will execute on processor $\pi_2$ over the interval $[A_6, A_6 + 8)$, which contradicts Lemma 7. Thus, in each subcase, we derived a contradiction. Thus, Equation 55 is impossible and Equation 54 must be true. ■

**Lemma 22** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then*

$$(A_5 < A_6 - 2) \vee (A_6 + 4 < A_5) \tag{56}$$

**Proof:** We prove the lemma by contradiction; that is, we will assume that there is not sufficient idle time for $J_6$ in $S'_I$ and

$$A_6 - 2 \leq A_5 \leq A_6 + 4. \tag{57}$$

However, we will show that the Equation 57 leads to a contradiction.

Our argument is based on a case analysis of the possible relative values of $A_6$ and $t_{start}$ (under the constraint of Equation 57). First, observe that Lemma 15 states that $[A_5, A_5 + 6) \subset [t_{start}, t_{start} + 8 - \alpha)$; thus, the following two inequalities are true:

$$t_{start} \leq A_5, \tag{58}$$

and

$$A_5 + 6 \leq t_{start} + 8 - \alpha, \tag{59}$$

Our case analysis contains three major cases (with several subcases). We will show in each case a contradiction arises. The three major cases are:

**Case I)** $A_6 - 2 \leq t_{start} \leq A_6 + 2 + \alpha$;

**Case II)** $t_{start} < A_6 - 2$; or

**Case III)** $A_6 + 2 + \alpha < t_{start}$.

Below is the proof of contradiction for each major case.
**Analysis for Case I):** This case directly contradicts Lemma 21.
**Analysis for Case II):** By assumption of Case II and Equation 59,

$$\begin{aligned} & A_5 - 2 + \alpha \leq t_{start} < A_6 - 2 \\ \Rightarrow \quad & A_6 - 4 + \alpha \leq t_{start} < A_6 - 2 \end{aligned} \tag{60}$$

The last implication follows from the assumption of Equation 57.

Let $y \stackrel{\text{def}}{=} A_6 - 2 - t_{start}$. We may rewrite the expression $t_{start} + 8 - \alpha$ as $A_6 + 6 - \alpha - y$ and the interval $[t_{start}, t_{start} + 8 - \alpha)$ as $[t_{start}, A_6 + 6 - \alpha - y)$. From Case II and Equation 60, we may obtain the following bounds on $y$:

$$0 < y \leq 2 - \alpha. \tag{61}$$

Since $[A_6, A_6 + 6 - \alpha - y) \subset [t_{start}, t_{start} + 8 - \alpha)$, Lemma 15 implies that $[A_6, A_6 + 6 - \alpha - y)$ is a continuously busy interval for tasks $\tau^A \cup \{\tau_5\}$ on processor $\pi_1$. Note that $A_6 + 6 - \alpha - y \geq A_6 + 4$ from Equation 61; so, the interval $[A_6, A_6 + 6 - \alpha - y)$ is non-empty. Since $A_6 + 6 - \alpha - y$ equals $t_{start} + 8 - \alpha$, Lemma 16 implies that no job of $\tau^A \cup \{\tau_5\}$ executes in $[A_6 + 6 - \alpha - y, A_6 + 8 - y)$. By $y > 0$ (Equation 61), the interval $[A_6 + 8 - y, A_6 + 8)$ is also a non-zero length interval. We have, thus, partitioned the interval $[A_6, A_6 + 8)$ into three disjoint, non-zero-length intervals: $[A_6, A_6 + 6 - \alpha - y)$, which is continuously busy for $\tau^A \cup \{\tau_5\}$; $[A_6 + 6 - \alpha - y, A_6 + 8 - y)$ which is continuously idle for $\tau^A \cup \{\tau_5\}$; and $[A_6 + 8 - y, A_6 + 8)$.

The only jobs that execute on processor $\pi_2$ over $[A_6, A_6 + 8)$ are $J_2^1$, $J_2^2$, $J_3^1$, $J_3^2$ and $J_5$, by Lemmas 8, 7, and 5. Since $[A_6, A_6 + 6 - \alpha - y)$ equals $[t_{start}, t_{start} + 8 - \alpha) \cap [A_6, A_6 + 8)$ and $[A_5, A_5 + 6) \subseteq [t_{start}, t_{start} + 8 - \alpha)$, Lemma 5 implies that $J_5$ must execute for some amount of time $\leq \alpha$ in the interval $[A_6, A_6 + 6 - \alpha - y)$. We consider the following subcase analysis based on the relative placement of the two jobs of $\tau_2$ and $\tau_3$. The subcases are:

*Subcase II.A)* $\tau_2$ has the scheduling windows of both $J_2^1$ and $J_2^2$ intersect with $[A_6, A_6 + 6 - \alpha - y)$;

*Subcase II.B)* $\tau_2$ has at most one job that intersects with $[A_6, A_6 + 6 - \alpha - y)$;

- *Sub-Subcase II.B1)* $\tau_3$ has the scheduling windows of both $J_3^1$ and $J_3^2$ intersect with $[A_6, A_6 + 6 - \alpha - y)$;
- *Sub-Subcase II.B2)* both $\tau_2$ and $\tau_3$ have at most one job that intersects with $[A_6, A_6 + 6 - \alpha - y)$.

For Subcase II.A, the interval between the scheduling windows of $J_2^1$ and $J_2^2$ must be completely contained in $[A_6, A_6 + 6 - \alpha - y)$ (i.e., $[A_2^1 + 1, A_2^2) \subset [A_6, A_6 + 6 - \alpha - y)$). The period and relative deadline parameter of $\tau_2$ ($p_2 = 5$ and $d_2 = 1$) imply that $A_2^2 - (A_2^1 + 1) \geq 4$. The length of $[A_6, A_6 + 6 - \alpha - y)$ is strictly less than $6 - \alpha$, since $y > 0$. Therefore, the total intersection between the scheduling windows of both $J_2^1$ and $J_2^2$ and the interval $[A_6, A_6 + 6 - \alpha - y)$ is strictly less than $2 - \alpha$. The remaining portion of the scheduling windows, of total length at least $\alpha$, for $J_2^1$ and $J_2^2$ must overlap with either $[t_{start}, A_6)$ or $[A_6 + 6 - \alpha - y, A_6 + 8 - y)$. Since $e_2 = d_2 = 1$, $J_2^1$ must be continuously executing over the interval $[A_2^1, A_2^1 + 1) \cap [t_{start}, A_6)$ (if non-empty), since $J_2^1$ completes by its deadline in $S'_I$; note that the execution of $J_1^1$ is outside the interval $[A_6, A_6 + 8)$. Similarly, $J_2^2$ must also be continuously executing over the interval $[A_2^2, A_2^2 + 1) \cap [A_6 + 6 - \alpha - y, A_6 + 8 - y)$. Corollary 2 implies that $J_2^2$ must execute over $[A_6, A_6 + 8)$ for a non-zero amount. Lemma 17 implies that processor $\pi_1$ is continuously busy over the intervals $[A_2^2, A_2^2 + 1) \cap [A_6 + 6 - \alpha - y, A_6 + 8 - y)$. However, by the argument at the beginning of Case II, $[A_6 + 6 - \alpha - y, A_6 + 8 - y)$ does not contain the execution of jobs of $\tau^A \cup \{\tau_5\}$ on processor $\pi_1$. Therefore, $[A_2^2, A_2^2 + 1) \cap [A_6 + 6 - \alpha - y, A_6 + 8 - y)$ is continuously busy on processor $\pi_1$ executing jobs of $\tau_2$ and $\tau_3$. Lemma 18 implies that processor $\pi_2$ does not contain the execution of jobs of $\tau_2$ and $\tau_3$ over $[A_2^2, A_2^2 + 1) \cap [A_6 + 6 - \alpha - y, A_6 + 8 - y)$. Since $J_2^2$ must be continuously executing over its scheduling window and $\pi_2$ does not execute jobs of $\tau_2$ over $[A_2^2, A_2^2 + 1) \cap [A_6 + 6 - \alpha - y, A_6 + 8 - y)$, $J_2^2$ must execute entirely on processor $\pi_1$ over $[A_2^2, A_2^2 + 1) \cap [A_6 + 6 - \alpha - y, A_6 + 8 - y)$. Finally, observe that since $J_2^2$ overlaps with $[A_6, A_6 + 6 - \alpha - y)$, then $A_2^2 < A_6 + 6 - \alpha - y \Rightarrow A_2^2 + 1 < A_6 + 7 - \alpha - y < A_6 + 8 - y$; hence, $\tau_2$ does not execute during $[A_6 + 8 - y, A_6 + 8)$. Thus, we have shown that during the total portion (of length $\geq \alpha$) that the scheduling windows of $J_2^1$ and $J_2^2$ do not overlap with $[A_6, A_6 + 6 - \alpha - y)$ (specifically, $[t_{start}, A_6)$), $\tau_2$ either executes outside of $[A_6, A_6 + 8)$ on processor $\pi_2$ or $\tau_2$ executes on processor $\pi_1$. Thus, for Case II.A, the most that $\tau_2$ can execute on processor $\pi_2$ in $S'_I$ over $[A_6, A_6 + 8)$ is at most $2 - \alpha$. However, this directly contradicts Lemma 8 which states that $\tau_2$ executes for strictly more than $2 - \alpha$ time units on processor $\pi_2$ over $[A_6, A_6 + 8)$.

For Sub-Subcase II.B1, both $J_3^1$ and $J_3^2$ intersect with the interval $[A_6, A_6 + 6 - \alpha - y)$; thus, the interval between the scheduling windows of $J_3^1$ and $J_3^2$ must be completely contained in $[A_6, A_6 + 6 - \alpha - y)$ (i.e., $[A_3^1 + 2, A_3^2) \subset [A_6, A_6 + 6 - \alpha - y)$). The period and relative deadline parameter of $\tau_3$ ($p_3 = 6$ and $d_3 = 2$) imply that $A_3^2 - (A_3^1 + 2) \geq 4$. By reasoning similar to previous subcase above, the total intersection between the scheduling windows of $J_3^1$ and $J_3^2$ is strictly less than $2 - \alpha$. Thus, the most that $\tau_3$ can execute on processor $\pi_2$ over the interval $[A_6, A_6 + 6 - \alpha - y)$ is strictly less than $2 - \alpha$. $\tau_3$ cannot execute over $[A_6 + 6 - \alpha - y, A_6 + 8 - y)$, since if $J_3^2$ overlaps with $[A_6 + 6 - \alpha - y, A_6 + 8 - y)$ then Lemma 17 implies $\pi_1$ would be continuously busy executing $\tau_2$ or $\tau_3$ over $[A_3^2, A_3^2 + 2) \cap [A_6 + 6 - \alpha - y, A_6 + 8 - y)$. Lemma 18 implies that no jobs of $\tau_2$ or $\tau_3$ execute on $\pi_2$ over such an interval. Furthermore, observe that since $J_3^2$ overlaps with $[A_6, A_6 + 6 - \alpha - y)$, then $A_3^2 < A_6 + 6 - \alpha - y \Rightarrow A_3^2 + 2 < A_6 + 8 - \alpha - y < A_6 + 8 - y$; hence, $\tau_3$ does not execute during $[A_6 + 8 - y, A_6 + 8)$. Thus, for Sub-Subcase II.B1, the most that $\tau_3$ can execute on processor $\pi_2$ in $S'_I$ over $[A_6, A_6 + 8)$ is strictly less than $2 - \alpha$ which contradicts Lemma 7.

For Sub-Subcase II.B2, we have at most one job of each $\tau_2$ and $\tau_3$ that intersect with $[A_6, A_6 + 6 - \alpha - y)$. Notice that $J_2^2$'s scheduling window does not intersect with $[A_6, A_6 + 6 - \alpha - y)$. If $J_2^2$ intersects $[A_6, A_6 + 6 - \alpha - y)$, then $J_2^1$'s scheduling window must also intersect $[A_6, A_6 + 6 - \alpha - y)$ because by Lemma 8 $J_2^1$ intersects with $[A_6, A_6 + 6)$ and $A_2^1 + 5 \leq A_2^2$; however, this contradicts the assumption of Sub-Subcase II.B2. Similarly, it may be shown by identical reasoning that $J_3^2$'s scheduling window does not intersect with $[A_6, A_6 + 6 - \alpha - y)$. By Lemma 17 and 18, neither $J_2^2$ nor $J_3^2$ can execute on $\pi_2$ in schedule $S'_I$ during the interval $[A_6 + 6 - \alpha - y, A_6 + 8 - y)$. Thus, in this subcase, the only times during which $J_2^2$ or $J_3^2$ may execute on $\pi_2$ over $[A_6, A_6 + 8)$ is during the subinterval $[A_6 + 8 - y, A_6 + 8)$. However the length of the interval is at most $2 - \alpha$ by Equation 61. So, $J_2^2$ and $J_3^2$ contribute at most $2 - \alpha$ execution on $\pi_2$ over $[A_6, A_6 + 8)$. $J_2^1$ and $J_3^1$ contribute at most one unit on $\pi_2$ over $[A_6, A_6 + 8)$. Finally, $J_5$ contributes at most $\alpha$ units on $\pi_2$ over this interval. Thus, $\sum_{\tau_i \in \tau^{\text{example}} - \{\tau_6\}} W_i(S'_I, \pi_2, A_6, A_6 + 8) \leq 4$. In this case, $J_6$ could have completed its execution entirely on processor $\pi_2$. Thus, in each subcase, we derived a contradiction to our assumption of insufficient idle time for $J_6$.

**Analysis for Case III):** This case is exactly symmetric to Case II.

In each major case, we achieve a contradiction to our assumption that $J_6$ could not execute completely in $S'_I$. Thus, Equation 57 must be false. The lemma follows. ■

### 5.3.4 Step 4: Construction of Schedule $S''_I$

By the previous section, we know that if $\tau_6$ cannot complete in schedule $S'_I$, then there exists a job of $\tau_6$ where there is insufficient time on both $\pi_1$ and $\pi_2$ to complete the job during the idle instants. As in the last section, let $J_6$ be any such job of $\tau_6$ that cannot complete in its scheduling window with respect to the idle instants of $S'_I$. We now define a modified schedule $S''_I$ in which more of $\tau_5$'s execution on processor $\pi_2$ is moved out of the interval $[A_6, A_6 + 8)$. Lemma 5 implies that a job $J_5$ of $\tau_5$ exists that has a scheduling window that intersects with $[A_6, A_6 + 8)$. Lemma 22 implies that $J_5$'s scheduling window is not completely contained in $[A_6, A_6 + 8)$. The following are informal "rules" which we apply inductively at every time instant $t$ from $[0, \infty)$. A formal definition of $S''_I$ appears immediately after the informal description.

**Rule 0)** The schedule for processor $\pi_1$ is not changed from $S'_I$ to $S''_I$ (i.e, for all $t$, $S''_I(\pi_1, t) = S'_I(\pi_1, t)$).

**Rule 1)** The current job of $\tau_5$ has its execution moved to time $t$ on processor $\pi_2$ *if*:

a) there is a current job of $\tau_5$ at time $t$;

b) there is no current job of $\tau_6$ at time $t$;

c) no job was scheduled at time $t$ on processor $\pi_2$ in $S'_I$; and

d) the total execution of the current job of $\tau_5$ over its entire scheduling window on processor $\pi_1$ *plus* the total execution of the current job of $\tau_5$ on processor $\pi_2$ up until time $t$, is less than $\tau_5$'s execution requirement.

The purpose of this rule is to add new execution of $J_5$ to times when $[A_5, A_5 + 6)$ does not overlap with $[A_6, A_6 + 8)$ (when processor $\pi_2$ is idle at time $t$ and $J_5$ is eligible to continue executing).

**Rule 2)** Processor $\pi_2$ is idled at time $t$ *if*:

a) there is a current job of $\tau_5$ at time $t$;

b) there is no current job of $\tau_6$ at time $t$;

c) a job of $\tau_5$ executed at time $t$ on processor $\pi_2$ in schedule $S'_I$; and

d) the total execution of the current job of $\tau_5$ over its entire scheduling window on processor $\pi_1$ *plus* the total execution of the current job of $\tau_5$ on processor $\pi_2$ up until time $t$, already equals $\tau_5$'s execution requirement.

The purpose of this rule is to continue to idle processor $\pi_2$ at times $t$ when $[A_5, A_5 + 6)$ does not overlap with $[A_6, A_6 + 8)$ and $J_5$ has sufficient execution on processor $\pi_1$ over $[A_5, A_5 + 6)$ and execution on processor $\pi_2$ over $[A_5, t)$ to successfully complete.

**Rule 3)** This rule is used to move execution out of the intersection of the scheduling windows of jobs of $\tau_5$ and $\tau_6$. (Note the execution is added to the non-intersecting portion of the windows by Rule 1.) For this rule, we need to determine how much execution has already been moved, as well as determine the amount of execution of $\tau_5$ that could be moved forward in time. The specification of the third rule for $S''_I(\pi_2, t)$ is that processor $\pi_2$ is idled at time $t$ *if*:

a) there is a current job of $\tau_5$ at time $t$;

b) there is a current job of $\tau_6$ at time $t$;

c) a job of $\tau_5$ executed at time $t$ on processor $\pi_2$ in schedule $S'_I$; and

d) the total aggregation of the following expressions exceeds or equals $\tau_5$'s execution requirement:

i) total execution of the current job of $\tau_5$ over its entire scheduling window on processor $\pi_1$ in schedule $S''_I$;

ii) the total execution of the current job of $\tau_5$ on processor $\pi_2$ preceding $\tau_6$'s scheduling window (if any) in schedule $S''_I$;

iii) the total execution of current job of $\tau_5$ from the arrival of $\tau_6$'s job *plus* the total execution of $\tau_5$ in schedule $S'_I$ occurring after $\tau_6$'s scheduling window;

iv) the total idle time during the portion of $\tau_5$'s scheduling window that succeeds $\tau_6$'s scheduling window in schedule $S'_I$ (i.e., potential times to move $\tau_5$'s execution).

**Rule 4)** Finally, if none above rules' conditions are satisfied, then the schedule at time $t$ remains the same as in $S'_I$.

The schedule $S''_I$ is formally (and inductively) defined as follows.

$$S''_I(\pi_1, t) \stackrel{\text{def}}{=} S'_I(\pi_1, t)$$

$$S''_I(\pi_2, t) \stackrel{\text{def}}{=} \begin{cases} \varphi_5(I,t)\,, & \text{if } (r_5(I,t) < \infty) \text{ and } (r_6(t) = \infty) \text{ and } (S'_I(\pi_2,t) = \perp) \text{ and } (S'_I(\pi_1,t,\tau_5) = 0) \\ & \text{and } (W_5(S'_I, \pi_1, r_5(I,t), r_5(I,t)+6) + W_5(S''_I, \pi_2, r_5(I,t), t) < 2), \\ \perp\,, & \text{if } (r_5(I,t) < \infty) \text{ and } (r_6(t) = \infty) \text{ and } (S'_I(\pi_2,t,\tau_5) = 1) \text{ and} \\ & (W_5(S'_I, \pi_1, r_5(I,t), r_5(I,t)+6) + W_5(S''_I, \pi_2, r_5(I,t), t) = 2), \\ \perp\,, & \text{if } (r_5(I,t) < \infty) \text{ and } (r_6(I,t) < \infty) \text{ and } (S'_I(\pi_2,t,\tau_5) = 1) \text{ and} \\ & (W_5(S'_I, \pi_1, r_5(I,t), r_5(I,t)+6) + W_5(S''_I, \pi_2, r_5(I,t), \max\{r_5(I,t), r_6(I,t)\}) + \\ & W_5(S''_I, \pi_2, \max\{r_5(I,t), r_6(I,t)\}, t) + W_5(S'_I, \pi_2, \min\{d_5(I,t), d_6(I,t)\}, d_5(I,t)) \\ & + W_\perp(S'_I, \pi_2, \min\{d_5(I,t), d_6(I,t)\}, d_5(I,t)) \geq 2), \\ S'_I(\pi_2,t)\,, & \text{otherwise.} \end{cases} \tag{62}$$

Figure 5 shows two possible scenarios in which execution of $J_5$ on processor $\pi_1$ is moved from its original scheduled time instants in $S'_I$. It is straightforward to see that $S''_I$ remains valid.

**Lemma 23** *Schedule $S''_I$ is valid.*

**Proof:** It is easy to see that $S''_I$ is valid, as we are only moving execution of $\tau_5$ during $\tau_5$'s scheduling window. Furthermore, we ensure that $\tau_5$ does not execute concurrently with itself and that the total execution over $\tau_5$'s scheduling window does not exceed $\tau_5$'s execution requirement ($e_5 = 2$). ■

Before showing that schedule $S''_I$ can accommodate $J_6$'s execution, we prove a lemma regarding the conditions that must hold when a job of $\tau_5$ executes on processor $\pi_2$ in schedule $S''_I$.

**Lemma 24** *Let $t > 0$ be a time such that $r_5(I,t) < \infty$ and $r_6(I,t) = \infty$ (i.e., at time $t$ there is a current scheduling window for $\tau_5$, but not $\tau_6$). The current job of $\tau_5$, $\varphi_5(I,t)$, executes on processor $\pi_2$ at time $t$ in schedule $S''_I$ (i.e., $S''_I(\pi_2,t) = \varphi_5(I,t)$), if and only if, the following three conditions hold:*

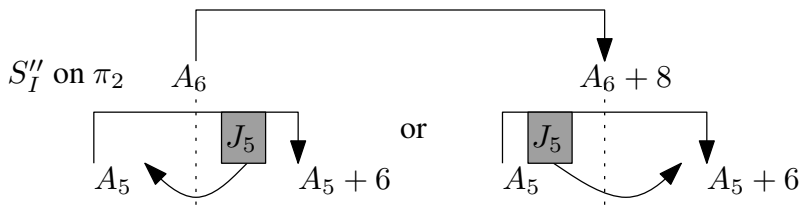



**Figure 5. The above image shows the two possible scenarios of moving $J_5$'s execution from the interval $[A_6, A_6 + 8)$. In the left scenario, $J_5$'s execution on processor $\pi_1$ in the interval $[A_6, A_6 + 8) \cup [A_5, A_5 + 6)$ is moved to the left in an available time instant on processor $\pi_1$ in the interval $[A_5, A_5 + 6) \setminus [A_6, A_6 + 8)$ that precedes $J_6$'s scheduling window. The movement of execution to left is achieved by application of Rule 1 followed by Rule 3. The right scenario shows the movement of execution to the right when $J_5$'s deadline is after $J_6$'s. Movement to the right is achieved by application of Rule 3 followed by Rule 1.**

**Condition 1:** *$\varphi_5(I, t)$ has not completed execution (i.e., $J_5$ has executed for a total of exactly two time units on $\pi_1$ over $[r_5(I, t), r_5(I, t) + 6)$ and $\pi_2$ over $[r_5(I, t), t)$). Formally, $W_5(S'_I, \pi_1, r_5(I, t), r_5(I, t) + 6) + W_5(S''_I, \pi_2, r_5(I, t), t) < 2$;*

**Condition 2:** *$\pi_1$ is not executing $\varphi_5(I, t)$ at time $t$ in schedule $S'_I$; and*

**Condition 3:** *$\pi_2$ is not executing a job of task $\tau_2$ or $\tau_3$ at time $t$ in schedule $S'_I$.*

**Proof:** The "if" direction is trivial; if each of the three conditions hold, observe that Rule 1's conditions are satisfied and $\varphi_5(I, t)$ is scheduled at time $t$ on processor $\pi_2$ in schedule $S''_I$. We will prove the "only if" direction by contradiction. That is, assume that $S''_I(\pi_2, t) = \varphi_5(I, t)$, but one of the three conditions is not true. Notice that if either Condition 1 or 2 is not true, the validity of schedule $S''_I$ (Lemma 23) will be violated. Specifically, if $\varphi_5(I, t)$ is scheduled at time $t$ on processor $\pi_2$, but has already executed two units on $\pi_1$ over $[r_5(I, t), r_5(I, t) + 5)$ and on $\pi_2$ over $[r_5(I, t), t)$, then we will execute for more than the execution requirement in $S''_I$. If $\varphi_5(I, t)$ is scheduled at time $t$ on processor $\pi_2$, but is already executing at time $t$ on processor $\pi_1$, then we will be executing concurrently with itself. Finally, if Condition 3 is not true, then either $\tau_2$ or $\tau_3$ was already executing at time $t$ in schedule $S'_I$. Observe that the definition of schedule $S''_I$ never moves execution of $\tau_2$ or $\tau_3$. So, we cannot concurrently execute a job of either $\tau_2$ or $\tau_3$ with $\varphi_5(I, t)$ on the same processor. Thus, in each case, we have shown that if any of the conditions is violated a contradiction arises. Therefore, if $\varphi_5(I, t)$ is scheduled at time $t$ on processor $\pi_2$ in schedule $S''_I$, then the above three conditions must hold. ■

We now show, for any job $J_6 \in I$ of task $\tau_6$ that cannot complete in schedule $S'_I$, $J_6$ is guaranteed to complete execution in $S''_I$. More formally, we show, in the following lemma, that there is sufficient space to execute $J_6$ entirely on processor $\pi_2$ in schedule $S''_I$ over $J_6$'s scheduling window.

**Lemma 25** *If $S'_I$ does not have sufficient idle time over $[A_6, A_6 + 8)$ to completely execute $J_6$, then*

$$\sum_{\tau_i \in \tau^{\text{example}} \setminus \{\tau_6\}} W_i(S''_I, \pi_2, A_6, A_6 + 8) \leq 4 \tag{63}$$

**Proof:** By Lemma 5, there exists a job $J_5 \in I$ of task $\tau_5$ with scheduling window such that $[A_5, A_5 + 6) \cap [A_6, A_6 + 8) \neq \emptyset$. From Lemma 22, exactly one of the following two expressions is true:

**Case I)** $A_5 < A_6 - 2$; or

**Case II)** $A_6 + 4 < A_5$.

**Analysis for Case I:** The inequality of Case I implies that $J_5$ arrives strictly earlier than two time units prior $J_6$'s arrival. Since $J_5$ and $J_6$ intersect, then $A_6 \leq A_5 + 6$. Therefore, $[A_6 - 2, A_6) \subset [A_5, A_5 + 6)$. There are two subcases to consider regarding the execution of $J_5$ over $[A_5, A_6)$.

*Subcase I.A)* $J_5$ completes $\alpha$ units of its execution on processor $\pi_2$ in $[A_5, A_6)$; or

*Subcase I.B)* $J_5$ does not complete $\alpha$ units of execution on processor $\pi_2$ in $[A_5, A_6)$.

For Subcase I.A, $J_5$ will not execute in the interval $[A_6, A_6 + 8)$; Lemma 5 states that $J_5$ executes for only $\alpha$ time on processor $\pi_2$ in $[A_5, A_5 + 6)$. Since $S''_I$ does not move execution of $\tau_2$ or $\tau_3$, exactly two jobs of both $\tau_2$ and $\tau_3$ execute in $[A_6, A_6 + 8)$ by Lemmas 8 and 7. The total execution requirement of these four jobs is at most four which implies Equation 63.

For Subcase I.B, note that Lemma 5 states that $J_5$ executes for $2 - \alpha$ on processor $\pi_1$. So, if $J_5$ does not complete $\alpha$ units of execution on processor $\pi_2$ in $[A_5, A_6)$ for schedule $S''_I$, then Condition 1 is never false for any $t \in [A_5, A_6)$. Therefore, by Lemma 24, whenever $J_5$ is not executing on processor $\pi_2$ over $[A_5, A_6)$, then either Condition 2 or 3 of Lemma 24 is false. By Lemma 5, $J_5$ executes on processor $\pi_2$ for $\alpha$ time units in $S'_I$ over $[A_5, A_5 + 6)$. Since $S''_I$ does not move additional execution of $J_5$ to $\pi_1$ from $\pi_2$, $J_5$ continues to execute for $\alpha$ time units in $[A_5, A_5 + 6)$ for $S''_I$. Thus, $J_5$ executes the remaining portion on processor $\pi_1$ for exactly $2 - \alpha$ time units. Hence, the most that $J_5$ could execute for in schedule $S''_I$ on processor $\pi_1$ over $[A_5, A_6)$ (and by extension subinterval $[A_6 - 2, A_6)$) is $2 - \alpha$. So, Condition 2 could be false for at most $2 - \alpha$ times in the interval $[A_6 - 2, A_6)$. The remaining $\alpha$ time in the interval $[A_6 - 2, A_6)$ must have either $J_5$ executing on processor $\pi_2$ or Condition 3 being false (i.e., either $\tau_2$ or $\tau_3$ are executing).

By Observation 1, the most $\tau_2$ can execute over $[A_6 - 2, A_6 + 8)$ (in any valid schedule) is two; similarly, the most $\tau_3$ can execute over $[A_6 - 2, A_6 + 8)$ is two. Lemma 5 and its period parameter ($p_5 = 100$) implies the most that $\tau_5$ could execute in $[A_6 - 2, A_6 + 8)$ is $\alpha$. Thus, the total execution of all jobs of $\tau^B$ over $[A_6 - 2, A_6 + 8)$ is $4 + \alpha$. By the preceding paragraph, at least $\alpha$ units of this execution on $\pi_2$ must occur over $[A_6 - 2, A_6)$, leaving at most four units to execute over $[A_6, A_6 + 8)$. Thus, Equation 63 is true for this subcase. We have shown that Equation 63 is true for all subcases of Case I

**Analysis for Case II:** Symmetric to Case I. ■

Theorem 2 immediately follows from the Steps outlined in Figure 3 and Lemma 25. That is, for any $I \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau^{\mathrm{example}})$, we can construct a valid schedule on two processors. Thus, by Definition 8, $\tau^{\mathrm{example}}$ is feasible on two processors.

## 6 Conclusions

In this article, we have seen that there exists a sporadic task system that is feasible upon a multiprocessor platform for which there does not exist an online multiprocessor algorithm that can successfully schedule every real-time instance generated by this task system. The existence of such a feasible task system implies that optimal online scheduling of sporadic and more general task systems is impossible for multiprocessor platforms. This article identified the feasible task system and proved that no online scheduling algorithm can successfully schedule all feasible instances.

The consequence of this negative result is far-reaching in that algorithms that are optimal for LL task systems no longer retain their optimality for small generalizations of the task model. Without optimality, it is not immediately clear what should be the theoretical basis for evaluating the effectiveness of a real-time multiprocessor scheduling algorithm for sporadic and more general task systems. The use of analytical techniques such as resource-augmentation [19] for identifying *near*-optimal online scheduling algorithms provide a potential metric for comparison of multiprocessor scheduling algorithms for general task systems.

## References


[1] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard Real-Time Scheduling: The Deadline Monotonic Approach. In *Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software*, pages 127–132, Atlanta, May 1991.

[2] T. Baker and M. Cirinei. A necessary and sometimes sufficient condition for the feasibility of sets of sporadic hard-deadline tasks. In *Proceedings of the IEEE Real-time Systems Symposium*, pages 178–187, Rio de Janeiro, December 2006. IEEE Computer Society Press.

[3] T. Baker and M. Cirinei. Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks. In *Proceedings of the 10th International Conference on Principles of Distributed Systems*, pages 62–75, Guadeloupe, December 2007.

[4] S. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems: The International Journal of Time-Critical Computing*, 24(1):99–128, 2003.

[5] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *Real-Time Systems: The International Journal of Time-Critical Computing*, 17(1):5–22, July 1999.

[6] S. Baruah, N. Cohen, G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15(6):600–625, June 1996.

[7] S. Baruah, R. Howell, and L. Rosier. Feasibility problems for recurring tasks on one processor. *Theoretical Computer Science*, 118(1):3–20, 1993.

[8] M. Dertouzos. Control robotics : the procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

[9] M. Dertouzos and A. K. Mok. Multiprocessor scheduling in a hard real-time environment. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, 1989.

[10] S. K. Dhall and C. L. Liu. On a real-time scheduling problem. *Operations Research*, 26:127–140, 1978.

[11] N. Fisher and S. Baruah. The feasibility of general task systems with precedence constraints on multiprocessor platforms. *Real-Time Systems*, 41(1):1–26, 2009.

[12] K. Hong and J. Leung. On-line scheduling of real-time tasks. In *Proceedings of the Real-Time Systems Symposium*, pages 244–250, Huntsville, Alabama, December 1988. IEEE.

[13] W. Horn. Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21:177–185, 1974.

[14] K. Jeffay, D. Stanat, and C. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the 12th Real-Time Systems Symposium*, pages 129–139, San Antonio, Texas, December 1991. IEEE Computer Society Press.

[15] A. N. Kolmogorov and S. V. Fomin. *Introductory Real Analysis*. Dover Publications, Inc., New York, 1970.

[16] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, 2:237–250, 1982.

[17] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[18] A. K. Mok. *Fundamental Design Problems of Distributed Systems for The Hard-Real-Time Environment*. PhD thesis, Laboratory for Computer Science, Massachusetts Institute of Technology, 1983. Available as Technical Report No. MIT/LCS/TR-297.

[19] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 140–149, El Paso, Texas, 4–6 May 1997.

[20] A. Srinivasan and J. Anderson. Optimal rate-based scheduling on multiprocessors. In *Proceedings of the 34th ACM Symposium on the Theory of Computing*, pages 189–198, May 2002.