The Multiprocessor Real-Time Scheduling of General Task Systems

by
Nathan Wayne Fisher

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2007

Approved by:
Sanjoy K. Baruah
James H. Anderson
Kevin Jeffay
Giuseppe Lipari
Ketan Mayer-Patel

Abstract

NATHAN WAYNE FISHER: The Multiprocessor Real-Time Scheduling of General Task Systems.
(Under the direction of Sanjoy K. Baruah.)

The recent emergence of multicore and related technologies in many commercial systems has increased the prevalence of multiprocessor architectures. Contemporaneously, real-time applications have become more complex and sophisticated in their behavior and interaction. Inevitably, these complex real-time applications will be deployed upon these multiprocessor platforms and require temporal analysis techniques to verify their correctness. However, most prior research in multiprocessor real-time scheduling has addressed the temporal analysis only of Liu and Layland task systems. The goal of this dissertation is to extend real-time scheduling theory for multiprocessor systems by developing temporal analysis techniques for more general task models such as the sporadic task model, the generalized multiframe task model, and the recurring real-time task model. The thesis of this dissertation is:

> *Optimal online multiprocessor real-time scheduling algorithms for sporadic and more general task systems are impossible; however, efficient, online scheduling algorithms and associated feasibility and schedulability tests, with provably bounded deviation from any optimal test, exist.*

To support our thesis, this dissertation develops feasibility and schedulability tests for various multiprocessor scheduling paradigms. We consider three classes of multiprocessor scheduling based on whether a real-time job may migrate between processors: *full-migration*, *restricted-migration*, and *partitioned*. For all general task

systems, we obtain feasibility tests for arbitrary real-time instances under the full- and restricted-migration paradigms. Despite the existence of tests for feasibility, we show that optimal online scheduling of sporadic and more general systems is impossible. Therefore, we focus on scheduling algorithms that have constant-factor approximation ratios in terms of an analysis technique known as *resource augmentation*. We develop schedulability tests for scheduling algorithms, earliest-deadline-first (EDF) and deadline-monotonic (DM), under full-migration and partitioned scheduling paradigms. Feasibility and schedulability tests presented in this dissertation use the workload metrics of *demand-based load* and *maximum job density* and have provably bounded deviation from optimal in terms of resource augmentation. We show the demand-based load and maximum job density metrics may be exactly computed in pseudo-polynomial time for general task systems and approximated in polynomial time for sporadic task systems.

Acknowledgments

My daily survival in the Ph.D. program required countless selfless acts of support, generosity, and time by people in my personal and academic life. This section is my humble attempt at acknowledging and thanking the people that have given so freely throughout my Ph.D. career and made this dissertation possible.

I am deeply grateful to Sanjoy K. Baruah, my advisor, for the guidance, support, respect, and kindness that he has shown me over the last four years. Sanjoy's mentoring, friendship, and collegiality enriched my academic life and have left a profound impression on how academic research and collaboration should ideally be conducted. While many students are fortunate to find a single mentor, I have been blessed with two; Jim Anderson has been a constant source of invaluable encouragement, aid, and expertise during my years at UNC. I am extremely thankful to the members of my dissertation committee: Kevin Jeffay has provided me with wise advice and support in both my research and career search; Giuseppe Lipari has graciously offered to serve on my committee from across the Atlantic and provided unique feedback, comments, and questions on multiprocessor scheduling; Ketan Mayer-Patel has offered insightful comments and ideas to my research and its effective presentation. Other faculty member who I owe gratitude for their support of my research or major Ph.D. milestones include: Prasun Dewan, Don Smith, Jack Snoeyink, and Mary Whitton. I am also grateful to the always helpful UNC Computer Science department staff; in particular, I owe special thanks to Sandra Neely, Janet Jones, and Tammy Pike.

I would like to thank all my research collaborators (in addition to Sanjoy and Jim) who have enhanced my enthusiasm and understanding of real-time systems: Ted Baker, Joël Goossens, Pascal Richard, and Thi Huyen Châu Nguyen. Special thanks goes to Marko Bertogna who visited UNC during the Fall semester of 2006 and whom I spent many enjoyable lunches and dinners discussing multiprocessor scheduling. I, in part, owe my first academic position outside graduation to the letters of recommendation written by Jim, Joël, Kevin, Pascal, Sanjoy, and Ted, in addition to Gerhard Fohler and Giorgio Buttazzo. Since the path through the Ph.D. program would be much more difficult without examples of success, I am indebted to Shelby Funk and Uma Devi who have given friendship, guidance, and advice as recent real-time system Ph.D. graduates. In addition, Mithun Arora, Aaron Block, Björn Brandenburg, Vasile Bud, John Calandrino, Philip Holman, Hennadiy Leontyev, Abhishek Singh, and Mengsheng Zhang, past and present students of the real-time systems group, have all given me hours of helpful discussion and constructive criticism. Other UNC graduate students outside the real-time group that have been especially supportive include (but are not limited to): Nico Galoppo, Russ Gayle, Sasa Junuzovic, Stephen Olivier, and Jeff Terrell.

My family and friends have been an unending source of love and inspiration throughout my Ph.D. career. My parents, Wayne and Joy Fisher, have offered unconditional understanding and encouragement. My sisters, Krista, Jessica, and Natasha, have kept me sane with their humor and understanding. My close friends in Chapel Hill, Evan Davis, Andrea Ford, and James and Katharine McClure have provided hours of enjoyable distraction from my work. Most importantly, my love, best friend, and daily companion, Marcy Renski has given me constant compassion and inspiration to give my best to achieve my goals while selflessly asking for little in return for herself. Marcy has endured my long trips away from home, my untidy habits

and occasional forgetfulness when busy on research, and many hours of practice talks only to motivate me to try even harder. Marcy's family have also been caring and supportive.

Contents

List of Figures

List of Tables

List of Abbreviations

DAG	Directed Acyclic Graph
DBF	Demand-Bound Function
DGMF	Distributed Generalized Multiframe
DM	Deadline Monotonic
DP	Dynamic Priority
EDF	Earliest-Deadline First
FFD	First-Fit Decreasing
FJP	Fixed Job-Priority
FTP	Fixed Task-Priority
GMF	Generalized Multiframe
ICD	Implantable Cardioverter Defibrillator
LCM	Least Common Multiple
LL	Liu and Layland
LLF	Least-Laxity First
PTAS	Polynomial-Time Approximation Scheme
PPTAS	Pseudo-Polynomial-Time Approximation Scheme
RBF	Request-Bound Function
RM	Rate Monotonic
SMP	Symmetric Shared-Memory Multiprocessor
UMA	Uniform Memory Access

Chapter 1

Introduction

Real-time systems are designed to satisfy notions of temporal correctness and predictability. In a real-time system, computations must occur by specified times. In our daily lives, we rely on systems that have underlying temporal constraints including avionic control systems, medical devices, network processors, digital video recording devices, and many other systems and devices. In each of these systems there is a potential penalty or consequence associated with the violation of a temporal constraint. For example, in a safety-critical system, a temporal violation can be life-threatening: a patient wearing an Implantable Cardioverter Defibrillator (ICD) is at risk of cardiac arrest if the device does not administer shocks to the heart in a timely fashion. In other (less critical) applications, violations of temporal constraints may result in a degradation in the quality-of-service experienced by the application user: a user listening to an MP3 file may experience audio jitter if the frames of the file are not decoded at a consistent rate. Regardless of the application, a well-designed real-time system should eliminate or minimize temporal constraint violations.

In a *hard real-time system*, the penalty for even a single temporal constraint violation is unacceptable. Typically, a hard real-time system associates a hard *deadline* with each system computation. For a hard real-time system to be temporally correct,

each computation must successfully complete prior to its deadline. The designer of a hard real-time system must verify that the system is correct prior to system run-time; that is, for any possible execution of the system, the designer must verify that each execution results in *all* deadlines being met. For all but the simplest systems, the number of possible execution scenarios is either infinite or prohibitively large. Therefore, exhaustive simulation or testing cannot be used to verify the temporal correctness of a hard real-time system. Instead, formal analysis techniques are necessary to ensure that the designed real-time systems are, by construction, provably temporally correct and predictable.

For a system to be proven temporally correct, three aspects of a real-time system must be specified:

1. *Real-Time Workload*: the computation produced by the real-time system that must complete prior to its deadline. In many real-time systems, the workload is modeled using the concept of a *recurring tasks*. A recurring task initiates, over time, the execution of sequential chunks of code called *jobs*. Once a job is initiated it must successfully complete its execution by an associated deadline. For a hard real-time system to be temporally correct, each job must complete by its deadline.

2. *Processing Platform*: the set of hardware resources upon which the jobs of the real-time workload are executed. The set of hardware resources includes the processor(s), memory, cache, processor/memory interconnect, etc. A *uniprocessor* platform consists of a single processor; a *multiprocessor* platform is comprised of a set of two or more processors.

3. *Scheduling Algorithm*: the algorithm that determines, at any time, which set of jobs execute on the processing platform.

Over the past three decades, the majority of research on real-time formal verification techniques has focused predominately on uniprocessor systems. Prior research that has addressed multiprocessor real-time systems has assumed a relatively simple task model for real-time workloads; specifically, most prior research has assumed that the set of jobs generated by any task is homogenous (i.e., the execution characteristics and deadline constraints of each job are identical) and that the deadline of any job coincides with the arrival of the next job of the same task. Unfortunately, such simple task models preclude the consideration of real-time applications that exhibit more complex behavior (e.g., tasks that generate heterogenous workloads) or dynamically change their computational requirements at run-time.

Furthermore, the need to support real-time systems on multiprocessor platforms has been brought to the forefront by the development of multicore architectures. With the current emergence of commercial systems such as Intel's Core 2 Duo and Quad processors or IBM's Cell multiprocessor and chip manufacturers' forecast of over 32 cores on a chip in the near future (Calandrino et al., 2007), the next generation of embedded and real-time hardware platforms will undoubtedly have the capability for parallel execution, increasing the need for multiprocessor real-time analysis. Unfortunately, as the previous paragraph points out, most techniques for temporal analysis of uniprocessor systems cannot be trivially extended to multiprocessor systems

The goal of this dissertation is *to increase the number of types of real-time systems that can be proven temporally correct upon a multiprocessor platform.* The achievement of this goal implies that more complex applications can now be proven temporally correct on multiprocessor systems; ultimately, the realization of this goal facilitates the leveraging of more powerful multiprocessor systems by complex real-time applications that previously could only be temporally verified on uniprocessor systems. We broaden the scope of analyzable real-time systems by considering very

general task models that allow tasks to generate heterogeneous workloads; additionally, we remove some restrictive assumptions of the simpler model. For real-time systems that may be modeled by the more general models, we develop analytical techniques for formally verifying the temporal correctness of these systems upon multiprocessor platforms. Furthermore, we show that our proposed analytical techniques have bounded deviation from any "hypothetically" optimal verification technique.

The remainder of this chapter formally introduces the concepts and terms used throughout this dissertation. Section 1.1 formally describes models of real-time workload. Section 1.2 introduces the processing platforms considered. Section 1.3 formalizes, categorizes, and discusses various online multiprocessor scheduling algorithms. Section 1.4 more concretely introduces concepts used in formal verification of real-time systems. Section 1.5 explicitly details the contributions of this dissertation. Section 1.6 outlines the overall structure of this document.

1.1 Real-Time Workload Models and Assumptions

Throughout this dissertation, 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 *active* at time t if $t \in [A_i, A_i + D_i)$ and J_i has unfinished execution.

We denote a real-time *instance* I as a finite or infinite collection of jobs $I = \{J_1, J_2, \ldots\}$. Unless otherwise specified, we will assume that jobs are indexed in order

of their arrival-time (i.e., for $J_i, J_j \in I$: $i < j$ if $A_i < A_j$). $\mathcal{F}(I)$ 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 1.1 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 $0 \leq z \leq 2$. ■

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 repeat 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 τ_i specified according to the model M. For every execution of the system, τ_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 throughout this dissertation. Every system execution of task system τ will result in

the generation a real-time instance I. We will denote the set of real-time instances that τ can legally generate as $\mathcal{I}^{\mathrm{M}}(\tau)$. Based on the real-time instances that τ generates, we can classify τ as either *completely specified* or *partially-specified*. We now discuss the difference between these two types of systems.

1.1.1 Completely-Specified Recurrent Task Systems

If the arrival-time and deadline parameters of each job $J_i \in I$ can be determined prior to system run-time, τ is a completely-specified task system. Typically, completely-specified task systems are appropriate for applications that have completely predictable executions and do not exhibit dynamic behavior. For example in an avionic control system, the control system will sample and process the pilot's input command at strict periodic intervals (e.g., see (Kirsch et al., 2002)). A strict rate is required to ensure that flight control response does not degrade. A completely-specified system is sometimes called a *concrete* system (Jeffay et al., 1991).

1.1.1.1 Periodic Task Systems

The *periodic task model* (Liu and Layland, 1973) allows the specification of homogenous sets of jobs that recur at strict periodic intervals. A periodic task τ_i is specified by a three tuple (o_i, e_i, p_i): o_i is the *offset* of the first job generated by τ_i from system start time; e_i is the *worst-case execution time* of any job generated by τ_i; and p_i is the *period* or inter-arrival time between successive jobs of τ_i. The set of jobs generated by a periodic task τ_i with worst-case possible execution times is $\mathcal{J}^{\mathrm{P}}_{\mathrm{WCET}}(\tau_i) \stackrel{\text{def}}{=} \{(o_i, e_i, o_i + p_i), (o_i + p_i, e_i, o_i + 2p_i), (o_i + 2p_i, e_i, o_i + 3p_i), \ldots\}$. Figure 1.1 illustrates the jobs generated by τ_i. Let $I_{\mathrm{WCET}} = \bigcup_{\tau_i \in \tau} \mathcal{J}^{\mathrm{P}}_{\mathrm{WCET}}(\tau_i)$; then $\mathcal{I}^{\mathrm{P}}(\tau) \equiv \mathcal{F}(I_{\mathrm{WCET}})$ is the set of real-time instances that can be generated by theperiodic task system τ.

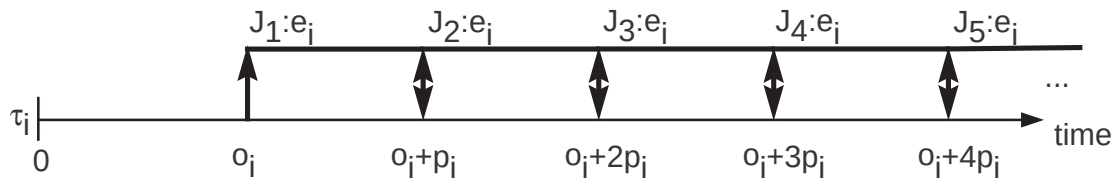


Figure 1.1: Jobs generated by periodic task τ_i. The first job arrives at time o_i. Thereafter, successive jobs arrive every p_i time units. The activation period of the k'th job of τ_i is the interval $[o_i + (k-1)p_i, o_i + kp_i)$. The "$J_i : e_i$" above each job indicates that job J_i must execute for e_i time units during its scheduling window.

Example 1.2 Consider a periodic task $\tau = \{\tau_1 = (0, 2, 4), \tau_2 = (5, 3, 10)\}$. The set of jobs generated by τ_1 with worst-case execution times is $\mathcal{J}^{\mathrm{P}}_{\mathrm{WCET}}(\tau_1) = \{(0, 2, 4), (4, 2, 4), (8, 2, 4), \ldots\}$; for τ_2, $\mathcal{J}^{\mathrm{P}}_{\mathrm{WCET}}(\tau_2) = \{(5, 3, 10), (15, 3, 10), (25, 3, 10), \ldots\}$. ■

1.1.2 Partially-Specified Recurrent Task Systems

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 such as periodic task systems are incapable of handling changes in real-time workloads because of the restrictive constraint that jobs must arrive at strict periodic intervals; for systems where the arrival times between jobs change dynamically (e.g., packets in a network), the periodic task model may not be appropriate. To overcome the fragile and inflexible nature of completely-specified task systems, a designer may instead consider partially-specified tasks systems. A partially-specified task system is sometimes referred to as *non-concrete* (Jeffay et al., 1991).

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 sys-

tem may legally generate infinitely many different real-time instances, each of which satisfies the constraints placed on 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 τ specified in model M there exists a task system τ' 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 τ that can be specified in task model M, there is a task system τ' specified in task model M' that can generate the same real-time instances as τ. The concept of generalizing a model will be made clearer in the remainder of this subsection.

In this subsection, we will introduce several increasingly general models for partially-specified task systems: the sporadic task model with implicit deadlines (Liu and Layland task model), general sporadic task model with explicit deadlines, and the recurring real-time task model. These increasingly general task models can be used to represent more complex applications than the restrictive periodic task model. After introducing the increasingly general models, we discuss the relationship between the various task models.

1.1.2.1 Sporadic Task Systems with Implicit Deadlines (Liu and Layland (LL) Task Model)

The sporadic task model with implicit deadlines (hereafter, referred to as the Liu and Layland (LL) task model (Liu and Layland, 1973)) removes the restrictive assumption of the periodic task model that jobs of a task are generated at strict periodic intervals (using the generalization discussed in (Mok, 1983)). In addition, an offset parameter is

not specified for LL tasks. The behavior of a LL task τ_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 τ_i. The p_i parameter indicates the *minimum inter-arrival time* between successive jobs of τ_i (note p_i denoted the *exact* inter-arrival time for periodic tasks). Let $\mathcal{J}^{\text{LL}}_{\text{WCET}}(\tau_i)$ be a collection of real-time instances that are jobs generated by LL task τ_i satisfying the minimum inter-arrival constraint and requiring the worst-case possible execution time; i.e., I_{τ_i} is a member of $\mathcal{J}^{\text{LL}}_{\text{WCET}}(\tau_i)$ if and only if for all $J_k \in I_{\tau_i}$ where $k > 0$ (recall that the jobs are indexed in order of non-decreasing arrival time) the following constraints are satisfied:

$$(E_k = e_i) \wedge (D_k = p_i) \wedge (A_{k+1} - A_k \geq p_i). \tag{1.1}$$

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

$$\mathcal{I}^{\text{LL}}_{\text{WCET}}(\tau) \stackrel{\text{def}}{=} \left\{ \bigcup_{i=1}^{n} I_{\tau_i} \;\middle|\; (I_{\tau_1}, I_{\tau_2}, \ldots, I_{\tau_n}) \in \prod_{i=1}^{n} \mathcal{J}^{\text{LL}}_{\text{WCET}}(\tau_i) \right\}. \tag{1.2}$$

Thus, the set of real-time instances generated by LL task system τ is

$$\mathcal{I}^{\text{LL}}(\tau) = \bigcup_{I_j \in \mathcal{I}^{\text{LL}}_{\text{WCET}}(\tau)} \mathcal{F}(I_j). \tag{1.3}$$

Figure 1.2 shows an example release for LL task τ_i. The following example illustrates the increase in flexibility in considering the LL task model over the periodic task model.

Example 1.3 Consider a LL task system with parameters similar to the task system of Example 1.2: $\tau = \{\tau_1 = (2, 4), \tau_2 = (3, 10)\}$. Examples of sets of jobs in $\mathcal{J}^{\text{LL}}_{\text{WCET}}(\tau_1)$ are $\{(0, 2, 4), (4, 2, 4), (8, 2, 4), \ldots\}$, $\{(0, 2, 4), (5, 2, 4), (9, 2, 4)\}$, and $\{(0, 2, 4), (6, 2, 4),$

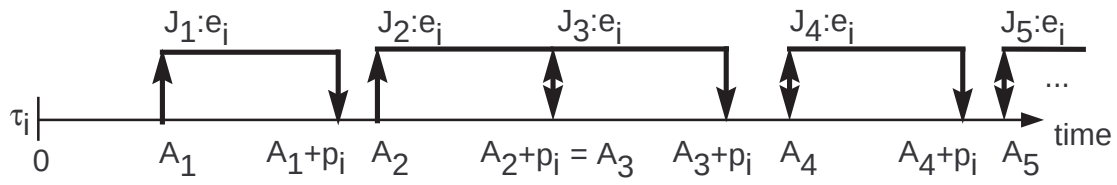


Figure 1.2: Jobs generated by LL task τ_i. The first job of τ_i can arrive at any time; in this figure, the first job arrives at time A_1. Thereafter, successive jobs arrivals must be separated by at least p_i time units. The $J_i : e_i$ above each job indicates that job J_i must execute for e_i time units during its scheduling window.

$(10, 2, 4), \ldots\}$; examples of sets of jobs in $\mathcal{J}^{\text{LL}}_{\text{WCET}}(\tau_2)$ are $\{(0, 3, 10), (10, 3, 10), (20, 3, 10), \ldots\}$, $\{(1, 3, 10), (15, 3, 10), (25, 3, 10), \ldots\}$, and $\{(5, 3, 10), (15, 3, 10), (25, 3, 10), \ldots\}$. Note that $\mathcal{J}^{\text{P}}_{\text{WCET}}((o_i, e_i, p_i) = (0, 2, 4))$ is an member of $\mathcal{J}^{\text{LL}}_{\text{WCET}}(\tau_1)$ and $\mathcal{J}^{\text{P}}_{\text{WCET}}((o_j, e_j, p_j) = (5, 3, 10))$ is a member of $\mathcal{J}^{\text{LL}}_{\text{WCET}}(\tau_2)$ where $(0, 2, 4)$ and $(5, 3, 10)$ are the two periodic tasks from Example 1.2. ■

1.1.2.2 Sporadic Task Systems with Explicit Deadlines

The LL task model allows for flexibility in the job arrival times for a task τ_i; however, the model is still somewhat restrictive in forcing the deadline of each job generated by τ_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 with explicit deadlines* (Mok, 1983) (hereafter, simply referred to as the *sporadic task model*) which generalized the LL task model by adding a *relative deadline* parameter d_i to the specification for a task. The relative deadline parameter d_i indicates the offset of a job's deadline from the arrival time for any job generated by task τ_i. A sporadic task τ_i is specified by the three-tuple (e_i, d_i, p_i). Let $\mathcal{J}^{\text{S}}_{\text{WCET}}(\tau_i)$ be a collection of real-time instances that are jobs generated

by sporadic task τ_i satisfying the minimum inter-arrival constraint and requiring the worst-case possible execution time; i.e., I_{τ_i} is a member of $\mathcal{J}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_i)$ if and only if for all $J_k \in I_{\tau_i}$ where $k > 0$ (recall that the jobs are indexed in order of non-decreasing arrival time) the following constraints are satisfied:

$$(E_k = e_i) \wedge (D_k = d_i) \wedge (A_{k+1} - A_k \geq p_i). \tag{1.4}$$

(Note that the only difference from Equation 1.1 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

$$\mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau) \stackrel{\mathrm{def}}{=} \left\{ \bigcup_{i=1}^{n} I_{\tau_i} \;\middle|\; (I_{\tau_1}, I_{\tau_2}, \ldots, I_{\tau_n}) \in \prod_{i=1}^{n} \mathcal{J}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_i) \right\}. \tag{1.5}$$

Thus, the set of real-time instances generated by sporadic task system τ is

$$\mathcal{I}^{\mathrm{S}}(\tau) = \bigcup_{I_j \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau)} \mathcal{F}(I_j). \tag{1.6}$$

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.

1.1.2.3 Generalized Multiframe (GMF) Task Systems

Both the LL and sporadic task models are useful when the worst-case execution time, relative deadline, and minimum inter-arrival time of each job generated by a task is identical. However, for some real-time applications, the sequence of jobs produced

may not be homogenous. The *generalized multiframe task (GMF) model* [1] (Baruah et al., 1999) permits a task to be characterized as a repeating sequence of heterogenous real-time jobs.

A GMF task τ_i is comprised of a finite sequence of jobs (originally referred to as *frames*) that can be repeated (possibly infinitely). Let N_i be the number of jobs that comprises a sequence for τ_i. τ_i can be characterized by a three-tuple $\tau_i = (\vec{e_i}, \vec{d_i}, \vec{p_i})$ where $\vec{e_i}$, $\vec{d_i}$, and $\vec{p_i}$ are N_i-ary vectors. Vectors $\vec{e_i} = [e_0, e_1, \ldots, e_{N_i-1}]$, $\vec{d_i} = [d_0, d_1, \ldots, d_{N_i-1}]$, and $\vec{p_i} = [p_0, p_1, \ldots, p_{N_i-1}]$ represent (respectively) the worst-case execution requirement, relative deadline, and minimum separation parameter of each job in the sequence.

Let $\mathcal{J}^{\text{GMF}}_{\text{WCET}}(\tau_i)$ be a collection of real-time instances that are jobs generated by GMF task τ_i satisfying the minimum inter-arrival constraint and requiring the worst-case possible execution time; i.e., I_{τ_i} is a member of $\mathcal{J}^{\text{GMF}}_{\text{WCET}}(\tau_i)$ if and only if for all $J_k \in I_{\tau_i}$ where $k > 0$, the following constraints are satisfied:

$$(E_k = e_{(k-1) \bmod N_i}) \wedge (D_k = d_{(k-1) \bmod N_i}) \wedge (A_{k+1} - A_k \geq p_{(k-1) \bmod N_i}). \quad (1.7)$$

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

$$\mathcal{I}^{\text{GMF}}_{\text{WCET}}(\tau) \stackrel{\text{def}}{=} \left\{ \bigcup_{i=1}^{n} I_{\tau_i} \;\middle|\; (I_{\tau_1}, I_{\tau_2}, \ldots, I_{\tau_n}) \in \prod_{i=1}^{n} \mathcal{J}^{\text{GMF}}_{\text{WCET}}(\tau_i) \right\}. \quad (1.8)$$

Thus, the set of real-time instances generated by GMF task system τ is

[1] The generalized multiframe task model is a generalization of both the sporadic task model and another model known as the *multiframe task model* (Mok and Chen, 1996).

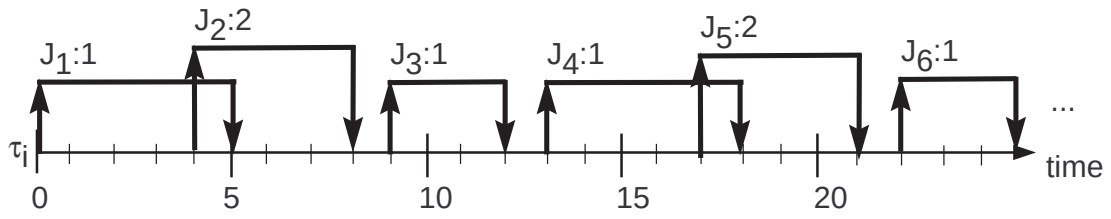


Figure 1.3: Jobs generated by GMF task τ_i from Example 1.4.

$$\mathcal{I}^{\mathrm{GMF}}(\tau) = \bigcup_{I_j \in \mathcal{I}^{\mathrm{GMF}}_{\mathrm{WCET}}(\tau)} \mathcal{F}(I_j). \tag{1.9}$$

Example 1.4 Consider the following GMF task $\tau_i \stackrel{\mathrm{def}}{=} ([1,2,1],[5,4,3],[4,5,4])$. A possible real-time instance $I_{\tau_i} \in \mathcal{J}^{\mathrm{GMF}}_{\mathrm{WCET}}(\tau_i)$ is $\{(0,1,5),(4,2,4),(9,1,3),(13,1,5),(17,2,4),\ldots\}$. This sequence corresponds to τ_i generating its first job at time zero, and successive jobs are generated as soon as legally allowable. Figure 1.3 illustrates this arrival sequence. Note that it is permissible for a job to arrive prior to the preceding job's deadline (i.e., two or more jobs may be in their scheduling window at a given time). ■

For any sporadic task system $\tau = \{\tau_1 = (e_1, d_1, p_1), \ldots, \tau_n = (e_n, d_n, p_n)\}$ we can represent the same task system in the GMF task model by the GMF task system $\tau' = \{\tau'_1 = ([e_1],[p_1],[p_1]), \ldots, \tau_n = ([e_n],[p_n],[p_n])\}$ (i.e., each vector is one-dimensional). It is easy to see that $\mathcal{I}^{\mathrm{S}}(\tau) = \mathcal{I}^{\mathrm{GMF}}(\tau')$; therefore, the GMF task model generalizes the sporadic task model.

1.1.2.4 Recurring Real-Time Task Systems

Each of the previous task models allow for the generation of sequences of jobs by a task: the LL and sporadic task models allow for a sequence of homogenous jobs from a task, and the GMF task model allows repeating sequences of heterogenous jobs. In a

sense, these models "fix" the relative sequential order of jobs during task specification. However, a real-time application may need to generate different sequences of jobs contingent upon the state of the system at run-time. Consider the following simple temperature control system for maintaining a system temperature between a `low-threshold` and `high-threshold`:

1 **repeat**
  ▷ Sample current temperature.
2 Generate sampling job J_S with execution requirement E_S and relative deadline D_S.
3 **if** (`temperature` < `low-threshold`) **then**
    ▷ Initiate heating mechanism.
4   Generate heating system control job J_H with execution E_H and relative deadline D_H.
5 **elseif** (`temperature` > `high-threshold`) **then**
    ▷ Initiate cooling mechanism.
6   Generate cooling system control job J_H with execution E_H and relative deadline D_H.
7 **end repeat**

The above temperature-control system generates a sequence of sample and heating/cooling jobs depending on the state of the system at each sample point. Obviously, the previously discussed recurrent task models cannot easily model the sequences produced by such a system.

(Baruah, 2003) introduced the *recurring real-time task model* to address such conditional behavior by real-time tasks. In the recurring real-time task model, a task τ_i is represented via a directed acyclic graph (DAG) with a unique source and unique sink vertex. A source vertex is a vertex with no incoming directed edges. A sink vertex has no outgoing edges. Let the DAG associated with τ_i be denoted by $G(\tau_i) = (\mathsf{Vertices}(\tau_i), \mathsf{Edges}(\tau_i))$ where $\mathsf{Vertices}(\tau_i)$ is a set of labels for the vertices of $G(\tau_i)$ and $\mathsf{Edges}(\tau_i) \subseteq \mathsf{Vertices}(\tau_i) \times \mathsf{Vertices}(\tau_i)$. Associated with each vertex $v \in \mathsf{Vertices}(\tau_i)$ is an execution requirement $e(v)$ and a relative deadline $d(v)$; the interpretation is that when τ_i generates a job associated with vertex v, it will have to complete at most $e(v)$ units of execution within $d(v)$ time units. Associated with each

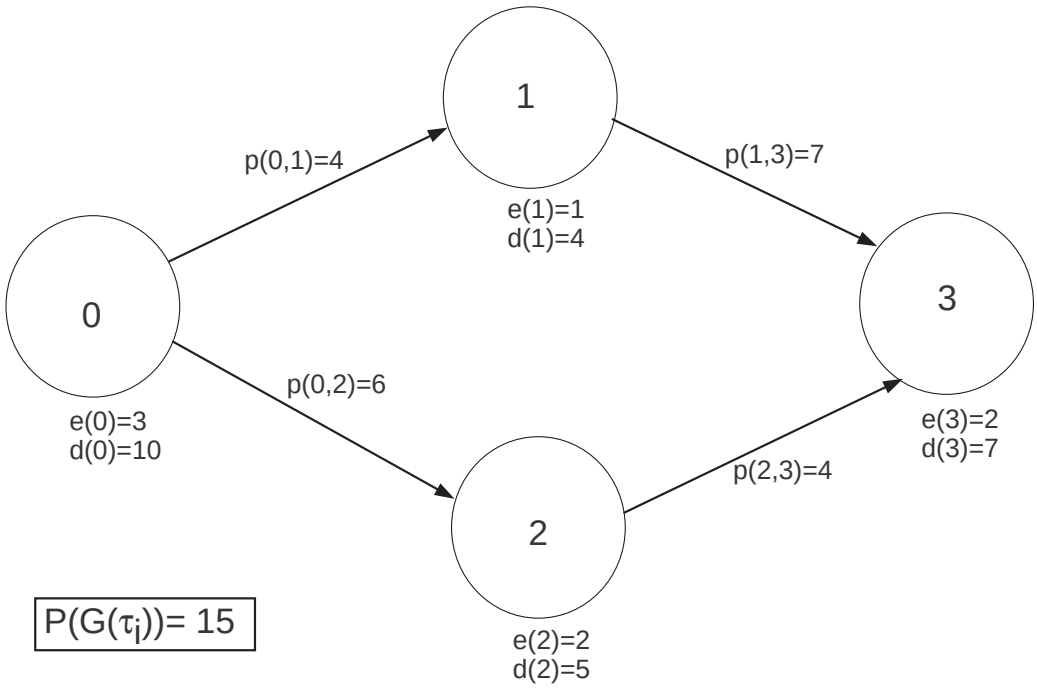


Figure 1.4: A recurring real-time task τ_i with four vertices.

edge $(u, v) \in \mathsf{Edges}(\tau_i)$ is a minimum separation $p(u, v)$, which represents the minimum time between the successive generation of jobs associated with vertices u and v. Finally, associated with the entire graph is a parameter $P(G(\tau_i))$, which represents the minimum time between generation of jobs corresponding to the source vertex (i.e., the jobs corresponding to the source vertex may not have their arrival times less than $P(G(\tau_i))$ time units apart). For any job J generated by τ_i, let $\mathsf{vertex}(J)$ be the label of the corresponding vertex in $G(\tau_i)$. Figure 1.4 illustrates an example specification of a recurring real-time task.

Let $\mathcal{J}^{\mathrm{R}}_{\mathrm{WCET}}(\tau_i)$ be a collection of real-time instances that are jobs generated by recurring real-time task τ_i satisfying the minimum inter-arrival constraints and requiring the worst-case possible execution time. I_{τ_i} is a member of $\mathcal{J}^{\mathrm{R}}_{\mathrm{WCET}}(\tau_i)$ if and only if for all $J_k \in I_{\tau_i}$ where $k > 0$, the following constraints are satisfied:

1. $E_k = e(\mathsf{vertex}(J_k))$.

2. $D_k = d(\mathsf{vertex}(J_k))$.

3. If $\mathsf{vertex}(J_k)$ is the sink, then J_{k+1} must also satisfy the following three constraints:

   a) $\mathsf{vertex}(J_{k+1})$ is the source vertex.

   (a) $A_{k+1} \geq A_k$ (i.e., a source vertex job cannot arrive before the previous sink vertex job).

   (b) For all source vertices J_ℓ ($\ell < k+1$), $A_{k+1} - A_\ell$ must be at least $P(G(\tau_i))$.

4. If $\mathsf{vertex}(J_k)$ is not the sink, then the following two constraints must be satisfied:

   (a) $(\mathsf{vertex}(J_k), \mathsf{vertex}(J_{k+1})) \in \mathsf{Edges}(\tau_i)$ (i.e., successive jobs must correspond to an edge in $G(\tau_i)$).

   (b) $A_{k+1} - A_k$ must be at least $p(\mathsf{vertex}(J_k), \mathsf{vertex}(J_{k+1}))$.

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

$$\mathcal{I}^{\mathrm{R}}_{\mathrm{WCET}}(\tau) \stackrel{\mathrm{def}}{=} \left\{ \bigcup_{i=1}^{n} I_{\tau_i} \;\middle|\; (I_{\tau_1}, I_{\tau_2}, \ldots, I_{\tau_n}) \in \prod_{i=1}^{n} \mathcal{J}^{\mathrm{R}}_{\mathrm{WCET}}(\tau_i) \right\}. \tag{1.10}$$

Thus, the set of real-time instances generated by recurring real-time task system τ is

$$\mathcal{I}^{\mathrm{R}}(\tau) = \bigcup_{I_j \in \mathcal{I}^{\mathrm{R}}_{\mathrm{WCET}}(\tau)} \mathcal{F}(I_j). \tag{1.11}$$

Example 1.5 Using the task specification of the task from Figure 1.4, the following is a possible real-time instance $I_{\tau_i} \in \mathcal{J}^{\mathrm{R}}_{\mathrm{WCET}}(\tau_i)$: $\{(0,3,10),\ (4,1,4),\ (11,2,7),\ (15,3,10),\ (21,2,5),\ (25,2,7),\ (30,3,10),\ \ldots\}$. The example sequence corresponds to the jobs of the "top path" (vertices 0, 1, and 3) being generated first, followed by jobs of the "bottom" path (vertices 1, 2, and 3). The jobs arrive as quickly as legally permitted. ■

For any GMF task $\tau_i = ([e_0, e_1, \ldots, e_{N_i-1}], [d_0, d_1, \ldots, d_{N_i-1}], [p_0, p_1, \ldots, p_{N_i-1}])$, we can represent the same task in the recurring real-time task model by the task τ'_i

where $\mathsf{Vertices}(\tau_i') = \{0, 1, 2, \ldots, N_i\}$ and $\mathsf{Edges}(\tau_i') = \{(\ell, \ell+1) | 0 \leq \ell < N_i\}$ (i.e., $G(\tau_i')$ is a unary tree with $N_i + 1$ vertices). Each vertex $v \in \{0, 1, \ldots, N_i - 1\}$ has the parameters $e(v) = e_v$ and $d(v) = d_v$. For vertex N_i, $e(N_i) = 0$ and $d(N_i) = 0$ (this is equivalent to a job with no work being produced and could be removed from the real-time instance with no side-effect). Each edge $(\ell, \ell+1) \in edges(\tau_i')$ has the parameter $p(\ell, \ell+1) = p_\ell$. The sequence period $P(G(\tau_i'))$ is set to zero; i.e., the jobs corresponding to the source vertex can be generated immediately after the generation of the preceding sink vertex job. It is straightforward to see that if jobs with zero execution are removed from $\mathcal{I}^{\mathrm{R}}(\tau')$ then $\mathcal{I}^{\mathrm{GMF}}(\tau) = \mathcal{I}^{\mathrm{R}}(\tau')$. Therefore, the recurring real-time task model generalizes the GMF task model. Figure 1.5 shows how the GMF task of Example 1.4 can be represented as a recurring real-time task.

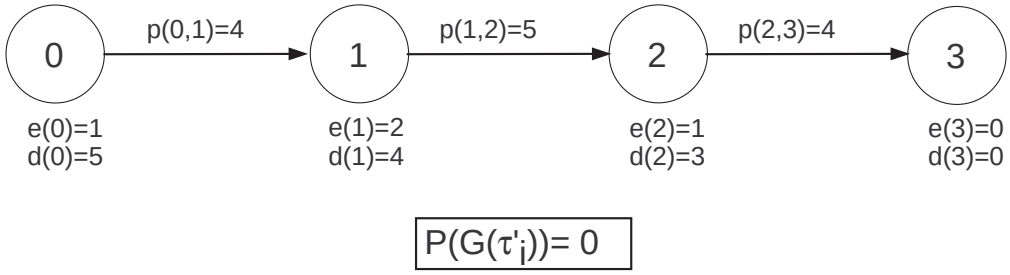


Figure 1.5: A recurring real-time task τ_i' that is equivalent to the GMF task $\tau_i \stackrel{\text{def}}{=} ([1, 2, 1], [5, 4, 3], [4, 5, 4])$ of Example 1.4.

1.1.2.5 Relationship Between Task Models

In this section, we have introduced increasingly general task models. Every generalization provides descriptive power to model increasingly complex behavior by real-time applications. Figure 1.6 illustrates the space of real-time instances that may be generated by partially-specified task systems in the various models described in this section. Table 1.1 summarizes the task models introduced in this section and briefly states the contribution of each model.

Task Model	Task Specification	Contribution
LL	Two-tuple	Minimum separation parameter
Sporadic	Three-tuple	Non-implicit relative deadline parameter
GMF	Three-tuple of N_i-ary vectors	Heterogeneous sequences of jobs
Recurring	DAG	Conditional job generation

Table 1.1: The above table summarizes the task models for partially-specified task systems introduced in this section. For each model, the task specification is informally described and a brief summary of the contribution of the task model is given.

We will see in Chapter 2 that most prior work in multiprocessor real-time systems has focused upon the simplest model discussed in this section: the LL task model. This dissertation instead focuses on expanding the types of systems that may be formally verified by considering the sporadic and more general task models. The most general multiprocessor scheduling results in this dissertation are valid for all real-time models described above. Section 1.5 more explicitly describes the contribution that developing formal temporal analysis for general task systems makes to multiprocessor real-time systems research.

1.2 Processing Platform

This dissertation 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; more specifically, each processor is identical in terms of architecture,

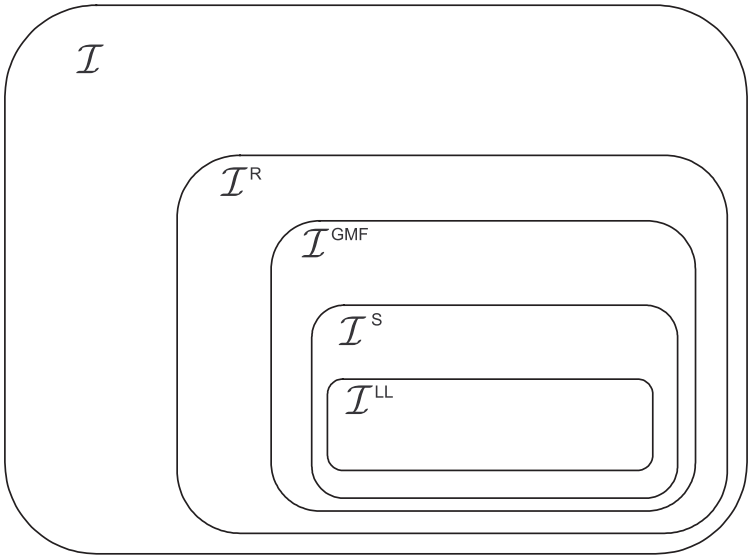


Figure 1.6: The space of real-time instances that can be generated by the models discussed in Section 1.1.2. $\mathcal{I}$ is the set of all real-time instances. $\mathcal{I}^{\mathrm{M}}$ is the set of all real-time instances than can be generated by a task system (with a finite number of tasks) specified in task model M. Note that $\mathcal{I} \supset \mathcal{I}^{\mathrm{R}} \supset \mathcal{I}^{\mathrm{GMF}} \supset \mathcal{I}^{\mathrm{S}} \supset \mathcal{I}^{\mathrm{LL}}$.

cache size and speed, I/O and resource access, and access time to shared memory (called Uniform Memory Access (UMA)). Figure 1.7 gives a high-level illustration of a possible layout of an identical multiprocessor platform. This type of multiprocessor layout is sometimes also called *symmetric shared-memory multiprocessor* (SMP). We denote the multiprocessor platform by Π and assume Π is comprised of m identical processors $\pi_1, \pi_2, \ldots, \pi_m \in \Pi$.

Recall from the beginning of this chapter that each job corresponds to the execution of a sequential segment of code by the processing platform. For each model introduced in the previous section (Section 1.1), 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. The process of determining the worst-case execution parameters is called *timing analysis*. Timing analysis must account for worst-case cache behavior, pipeline stalls, memory contention, mem-

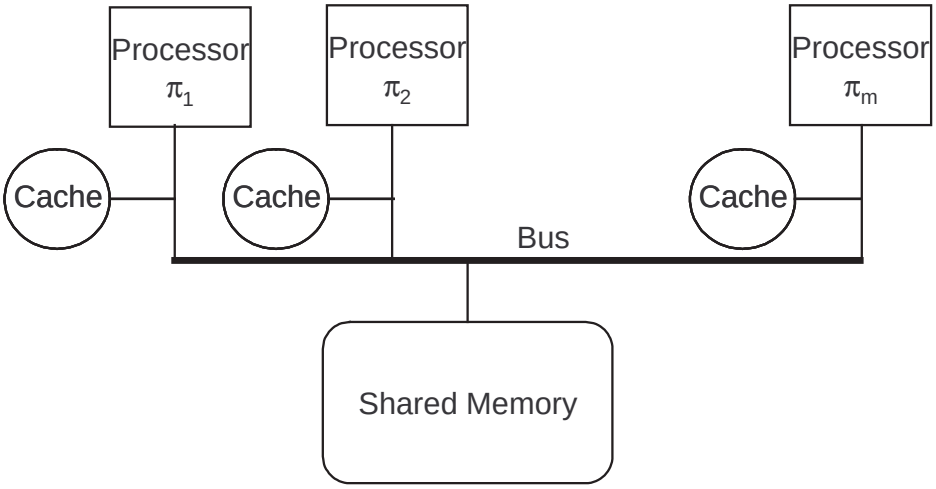


Figure 1.7: The layout of a symmetric shared-memory multiprocessor (SMP) platform.

ory access time, program structure, and worst-case execution paths. The analysis for determining the contribution to the worst-case execution time of each of these factors is dependent on the specific system and the program. Other factors that may increase the worst-case execution time are job *preemptions* (i.e., a job suspends while a different job executes and resumes execution at later time). The context switch, state saving, and scheduling-decision processing time by the platforms's operating system adds additional time to the job's execution requirement. Furthermore, if a job is allowed to *migrate* between processors during its scheduling window, there may be an added penalty of refreshing the cache of the processor to which the job is migrating. The preemption and migration execution costs are typically dependent on the processor architecture and the scheduling algorithm used. (Calandrino et al., 2006) determine the cost of preemption and migration for various multiprocessor scheduling algorithms on a Linux-based testbed. There are known techniques for accounting for these factors in the worst-case execution time parameter (see techniques described in (Devi, 2006; Baker and Baruah, 2007) for multiprocessor systems). In this dissertation, we will assume that the worst-case execution time of each task has already

been determined.

We will assume that each processor has unit-speed. We will assume that jobs are preemptable. The next section will discuss under what scenarios job migration between processors is allowed. Though a job may execute on different processors over its scheduling window, job-level parallelism is not permitted (i.e., a job may not execute concurrently with itself on two or more processors simultaneously); this assumption is not limiting, since we have defined a job to correspond to a sequential segment of code. Throughout this dissertation 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). Developing formal analysis techniques for general task systems that are not independent is the subject of current research and beyond the scope of this dissertation.

1.3 Real-Time Scheduling Algorithms

When executing a real-time application, the real-time scheduling algorithm must determine which active 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 Π. The interleaving of execution of I on Π 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 section, we discuss the classification of real-time multiprocessor scheduling algorithms. Section 1.3.1 gives some formal definitions for real-time scheduling algorithm concepts. Section 1.3.2 introduces a family of scheduling algorithms known as

priority-driven scheduling algorithms. Section 1.3.3 classifies multiprocessor scheduling algorithms based on the degree of migration permitted.

1.3.1 Notation

In this section and Section 1.4.1, we take a very formal approach in defining the concepts of real-time scheduling algorithms and formal verification techniques. In particular, we give mathematical notation and definitions for concepts that for most of the real-time literature a verbal definition sufficed. Our reasons for using a much more formal approach are twofold:

1. The more formal definitions allow us to reason about scheduling algorithms in very abstract terms. These abstractions will be used heavily in Chapter 4 and Appendix A.

2. The formal definitions make the connection between the concepts of scheduling algorithms, formal verification techniques, and real-time instance models explicit and unambiguous.

However, to reduce the burden on the reader in remembering notation, we will also provide a verbal description of each of the concepts introduced in these sections. We will use the verbal definitions when the more formal definitions are not required and the meaning is clear; the reader should refer back to the formal definitions of this chapter, if any confusion or ambiguity arises. The formal definitions will be used primarily in Chapter 4 and Appendix A.

As mentioned earlier, a schedule specifies the interleaving of execution of jobs of a real-time instance. That is, a schedule will indicate at any given time which job is executing on which processor. We can formally define the schedule S for real-time instance I as a function of the processor and time.

Definition 1.1 (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 Π.*

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 1.2 (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 π_k for schedule S_I. In other words,*

$$S_I(\pi_k, t, J_i) \stackrel{def}{=} \begin{cases} 1 \ , & \text{if } S_I(\pi_k, t) = J_i \\ 0 \ , & \text{otherwise.} \end{cases} \tag{1.12}$$

A scheduling algorithm makes decisions about the order in which jobs of a real-time instance should execute. If the real-time instance is specified prior to run-time or generated by a completely-specified task system, a scheduling algorithm can generate and store the schedule prior to run-time. This approach is called *static scheduling* or *table-driven scheduling* (see (Baker and Shaw, 1989) for an example static scheduler). For systems that are partially-specified or have schedules too large to store in a system's memory, an*online* algorithm is more appropriate. For any time t, an online real-time scheduling algorithm decides the set of jobs that will be executed on Π 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 dissertation focuses on deterministic online real-time scheduling algorithms.

At an abstract level, a real-time scheduling algorithm[2] $\mathcal{A}$ (either static or online) on platform Π is a higher-order function[3] from real-time instances to schedules over Π — i.e., $\mathcal{A} : \mathcal{I} \rightarrow \bigcup_{I \in \mathcal{I}} \mathbb{S}_{I,\Pi}$. 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 1.3 (Deterministic Online Scheduling Algorithm) *For any $I \in \mathcal{I}$, let $S_I^{\mathcal{A}}$ be the schedule produced by algorithm $\mathcal{A}$ for real-time instance I and platform Π. An online real-time scheduling algorithm must satisfy the following constraint: for all $I, I' \in \mathcal{I}$ and for all $t > 0$,*

$$(I_{\leq t} = I'_{\leq t}) \Rightarrow \left(\forall t'(0 \leq t' \leq t), \forall \pi_k \in \Pi :: S_I^{\mathcal{A}}(\pi_k, t') = S_{I'}^{\mathcal{A}}(\pi_k, t')\right). \tag{1.13}$$

1.3.2 Priority-Driven Scheduling Algorithms

A possible approach to the online scheduling of a real-time instance on a processing platform is to assign, at any given time t, each job J_i a priority $\rho(J_i, t)$ (which is assumed to be a real number). A *priority-driven* scheduling algorithm at each time t sorts the active jobs according to $\rho(J_i, t)$ (in non-decreasing order) and sched-

[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.

ules the highest-priority job(s) on the processing platform. In this section, we will describe priority-driven scheduling algorithms assuming a uniprocessor system; the next section (Section 1.3.3) will explain how these priority-driven algorithms will be used on multiprocessor platforms under different degrees of migration. Priority-driven scheduling algorithms differ in the manner that they assign priority to each job. In the following, we give three classifications of priority-driven scheduling algorithms. We follow the classification names used in (Baker and Baruah, 2007) ((Carpenter et al., 2003) provides a thorough overview of the types of priority-driven algorithms under slightly different terminology). The three major classes of priority-driven scheduling algorithms are *fixed task-priority* (FTP), *fixed job-priority* (FJP), and *dynamic priority* (DP).

1.3.2.1 Fixed Task-Priority (FTP) Scheduling Algorithms

In FTP scheduling, each task is assigned a fixed priority c, and each job generated by that task is assigned the same priority value. Thus, for all $J_k \in \mathcal{I}^{\mathrm{M}}(\tau)$ for any recurrent task model M, $\rho(J_k) = c$ for all $t \geq 0$.[4] For a recurrent task system with n tasks, there are n distinct priorities (one for each task). For FTP-scheduled systems, we will assume that tasks are indexed in decreasing order of priority; for $\tau \stackrel{\text{def}}{=} \{\tau_1, \ldots, \tau_n\}$, τ_1 has the highest priority and τ_n has the lowest priority. In general, the task-priority assignment can be determined by the system designer. However, there are two well-studied FTP-assignment algorithms for sporadic task systems: *rate monotonic* (RM) and *deadline monotonic* (DM).

§ **Rate Monotonic (RM).** For RM scheduling (Liu and Layland, 1973), each sporadic task τ_i is assigned priority equal to the inverse of its period: for all $J_k \in I_{\tau_i} (\in \mathcal{J}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_i))$, the priority $\rho(J_k)$ is equal to $1/p_i$.

[4] We omit the argument t from $\rho(\cdot)$ for FTP and FJP scheduling, since priority will not change over time for any job.

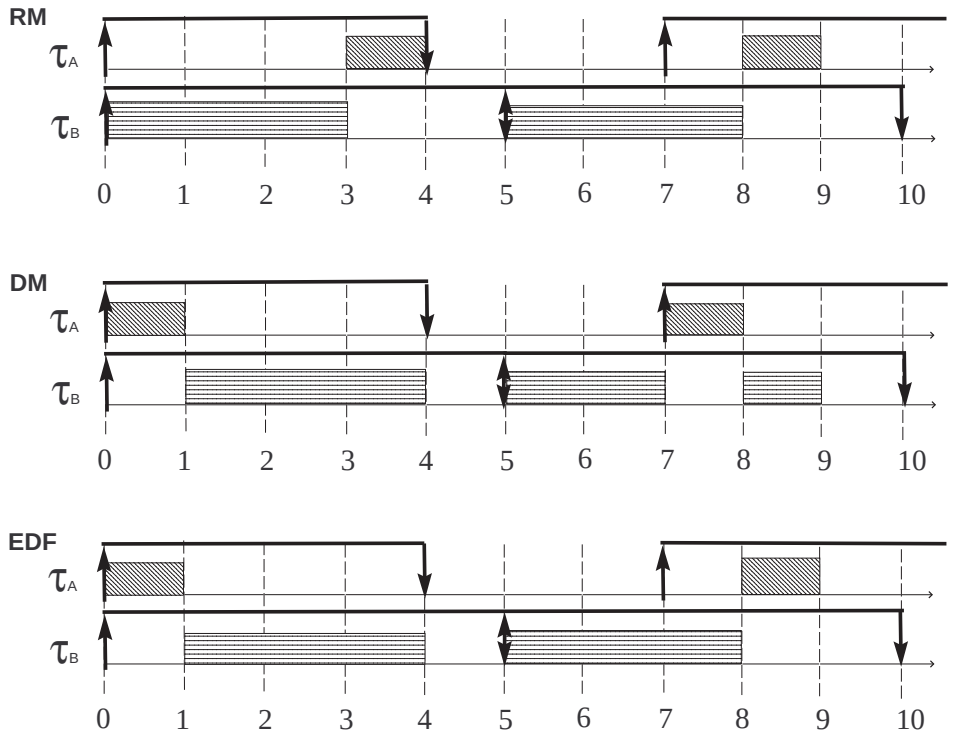


Figure 1.8: Possible schedules of RM, DM, and EDF for the task system of Example 1.6.

Example 1.6 Consider the sporadic task system $\tau \stackrel{\text{def}}{=} \{\tau_A, \tau_B\}$ where $\tau_A \stackrel{\text{def}}{=} (1, 4, 7)$ and $\tau_B \stackrel{\text{def}}{=} (3, 5, 5)$. The top schedule in Figure 1.8 gives a possible schedule for RM with respect to a legal job arrival sequence for τ. ■

§ **Deadline Monotonic (DM).** The DM scheduling algorithm (Leung and Whitehead, 1982) assigns to each sporadic task τ_i a priority equal to the inverse of its relative deadline parameter: for all $J_k \in I_{\tau_i} (\in \mathcal{J}^{\text{S}}_{\text{WCET}}(\tau_i))$, the priority $\rho(J_k)$ is equal to $1/d_i$. The middle schedule in Figure 1.8 gives a possible schedule for RM with respect to a legal job arrival sequence for τ from Example 1.6.

1.3.2.2 Fixed Job-Priority (FJP) Scheduling Algorithms

For FJP scheduling, the restriction that a task's jobs have identical priority is removed. Instead, each job J_k is assigned a single priority $\rho(J_k)$ that does not change.

The specific FJP scheduling algorithm determines the priority assignment for jobs. *Earliest-deadline first* (EDF) is a well known FJP scheduling algorithm.

§ **Earliest-Deadline First (EDF).** The EDF scheduling algorithm (Liu and Layland, 1973) assigns a priority to each job equal to the inverse of its absolute deadline. In other words, EDF schedules among the set of jobs with remaining execution the m jobs with the nearest deadline. For a recurrent task τ_i, the priority $\rho(J_k)$ of any job $J_k \in I_{\tau_i} (\in \mathcal{J}_{\text{WCET}}(\tau_i))$ equals $1/(A_k + D_k)$. The bottom schedule in Figure 1.8 gives a possible schedule for EDF with respect to a legal job arrival sequence for τ from Example 1.6.

1.3.2.3 Dynamic-Priority (DP) Scheduling Algorithms

The DP scheduling-algorithm classification is the most general. DP scheduling removes the restriction that a job priority does not change. A job J_k priority $\rho(J_k, t)$ can now vary over time. Examples of well-known DP scheduling algorithm include *least-laxity first* (LLF) and Pfair-based algorithms (Baruah et al., 1996; Baruah et al., 1995; Anderson and Srinivasan, 2004).

In this dissertation, we focus on FTP and FJP scheduling on multiprocessor platforms, primarily on the DM and EDF scheduling algorithms.

1.3.3 Degree of Migration

The allocation of real-time jobs to processors is another dimension of scheduling that may be used to classify real-time scheduling algorithms. Using the classification of (Carpenter et al., 2003), we consider three classes of multiprocessor scheduling algorithms: *partitioned scheduling*, *restricted-migration scheduling*, and *full-migration scheduling*.

1.3.3.1 Partitioned Scheduling

In partitioned scheduling, each recurrent task $\tau_i \in \tau$ is assigned a single processor $\pi_\ell \in \Pi$. The assignment of tasks to processors is typically done at system-design time. There are several algorithms and heuristics for assigning tasks to processors; Chapter 7 considers various partitioning algorithms for sporadic tasks. Once a task-processor assignment for τ_i to π_ℓ is determined, every job J_k generated by τ_i executes solely on processor π_ℓ. Let $\tau(\pi_k)$ denote the set of tasks assigned to processor π_k. The tasks of $\tau(\pi_k)$ are scheduled on processor π according to some uniprocessor scheduling algorithm. Figure 1.10 shows a high-level view of the partitioning approach.

Example 1.7 Consider the following three-task, sporadic task system $\tau \stackrel{\text{def}}{=} \{\tau_1 = (2, 3, 5), \tau_2 = (4, 8, 8), \tau_3 = (2, 4, 4)\}$. Let τ be scheduled upon two processors; the partition is $\tau(\pi_1) = \{\tau_1, \tau_2\}$ and $\tau(\pi_2) = \{\tau_3\}$. Figure 1.9(a) gives the partitioned schedule (with EDF used to schedule each individual processor) of a possible real-time instance generated by τ. ■

1.3.3.2 Full-Migration Scheduling

The least restrictive of the migration-based scheduling classifications is full-migration scheduling. In this classification, a job can halt its execution on one processor and resume execution on a different processor. The only major restriction for full-migration scheduling algorithms is that job-level parallelism is forbidden (i.e., a job may not execute concurrently with itself on two or more different processors). It is dependent on the scheduling algorithm whether task-level parallelism is permitted. Figure 1.11 gives a high-level overview of full-migration scheduling. Figure 1.9(b) illustrates the full-migration schedule for the task system of Example 1.7.

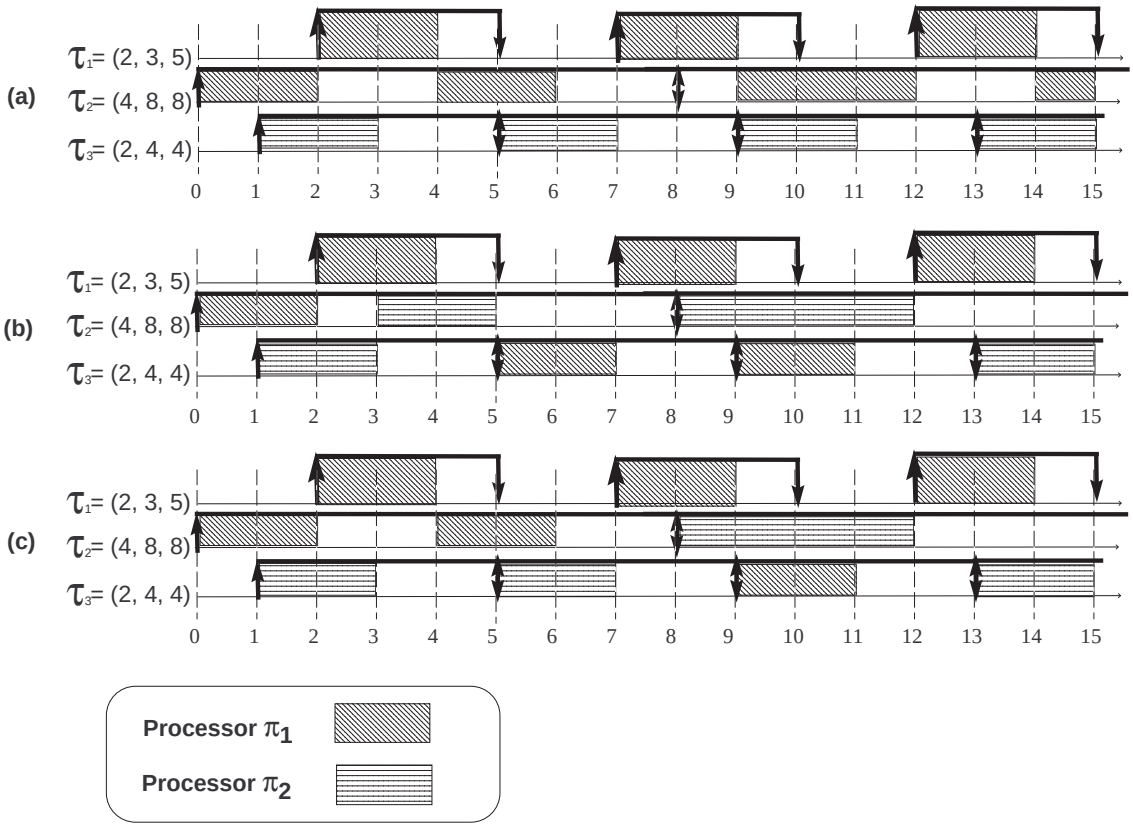


Figure 1.9: Example multiprocessor schedules for task system τ of Example 1.7 under EDF-scheduling and the different paradigms considered in this paper. **(a)** shows a partitioned schedule; **(b)** shows a full-migration schedule; and **(c)** shows the restricted-migration schedule.

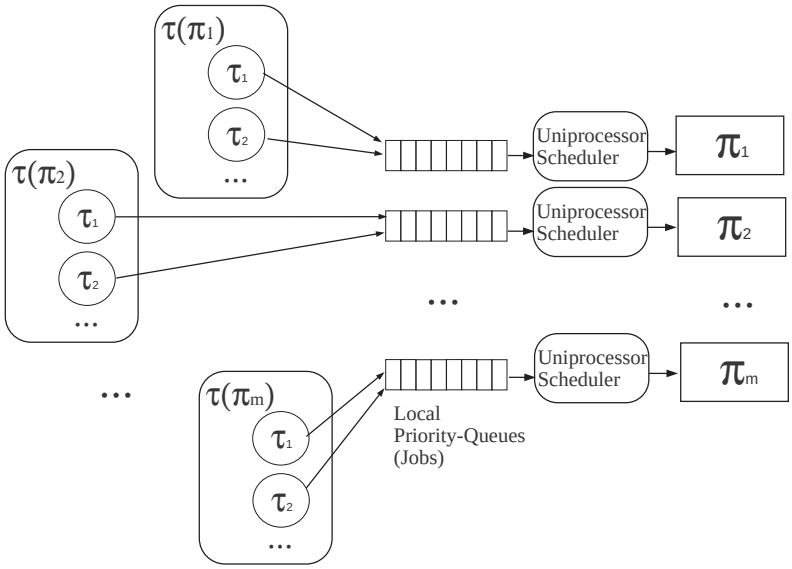


Figure 1.10: A high-level perspective of partitioned scheduling. Tasks are statically assigned to processors by a partitioning algorithm. Each task places generated jobs on the processor's local priority queue and a uniprocessor scheduling algorithm is used to schedule each processor.

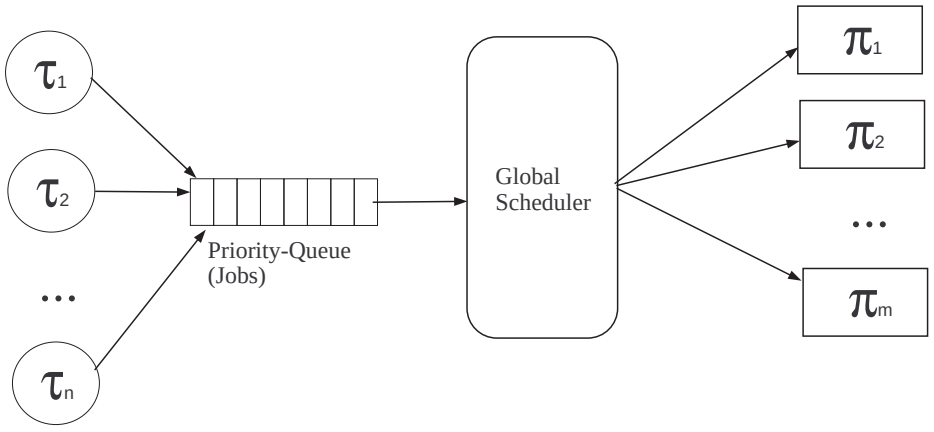


Figure 1.11: A high-level perspective of full-migration scheduling. Each job generated by a task is placed upon a global priority queue. A global scheduler decides what jobs execute on each processor at any given time.

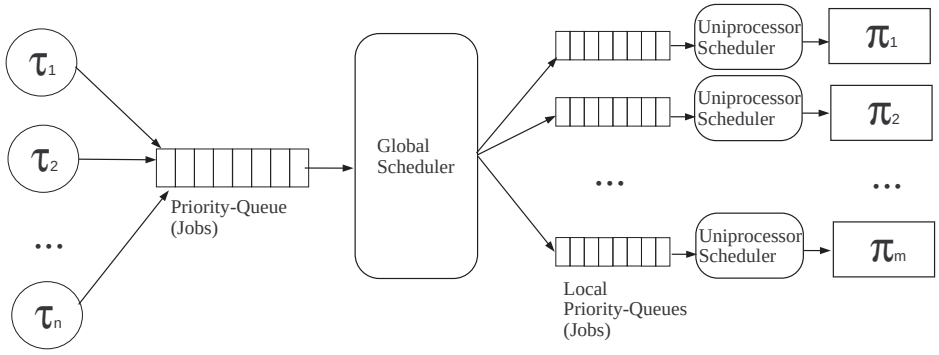


Figure 1.12: A high-level perspective of restricted-migration scheduling. There are essentially two-levels of schedulers. At run-time jobs generated by a task are placed on a global priority queue. A global scheduler assigns each job to a processor by placing it on the processor's priority queue. A uniprocessor scheduling algorithm is used to schedule each processor's assigned task.

1.3.3.3 Restricted-Migration Scheduling

Tasks are allowed to migrate between processors in a restricted-migration scheduling algorithm. Each job, however, must execute on only *one* processor. For partially-specified systems, the assignment of jobs to processors in restricted-migration scheduling is typically done online, since the specific job arrivals are not known *a priori*. For any real-time instance I, let $I(\pi_k) \subseteq I$ be the set of jobs assigned to processor $\pi_k \in \Pi$ by the restricted-migration scheduling algorithm. Like partitioned scheduling, the jobs of $I(\pi_k)$ (once assigned upon arrival) are scheduled using a uniprocessor scheduling algorithm. Like full-migration scheduling, it is dependent upon the algorithm whether task-level parallelism is allowed (i.e., two jobs of the same task executing on different processor concurrently). Figure 1.12 gives a high-level overview of restricted-migration scheduling. Figure 1.9(c) illustrates the full-migration schedule for the task system of Example 1.7.

1.4 Formal Verification of Real-Time Systems

As mentioned in the introduction, to ensure a real-time system is temporally correct it must be validated prior to system run-time using formal verification techniques. These formal verification techniques must ensure that for all legal executions of the system every real-time job generated by the system will meet its deadline on the processing platform. We consider two fundamental problems in formal verification for real-time systems: *feasibility analysis* and *schedulability analysis*. Feasibility analysis determines whether there always exists a "way" to schedule a system (irrespective of scheduling algorithm) meeting all deadlines. Schedulability analysis determines (with respect to a given scheduling algorithm) whether the scheduling algorithm will always meet all the system's deadlines.

The remainder of this section further introduces real-time formal verification techniques. Section 1.4.1 gives some notation that will be used to formally define feasibility and schedulability analysis. Section 1.4.2 discusses feasibility analysis for general task systems. Section 1.4.3 discusses schedulability analysis and related concepts for various real-time scheduling algorithms. Section 1.4.4 describes an approach, called *resource-augmentation analysis*, which is used to theoretically evaluate the effectiveness of real-time verification techniques.

1.4.1 Notation

This section gives formalism that will be used throughout this dissertation. 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 1.4 (Work Function) *$W(S_I, J_i, t_1, t_2)$ denotes the amount of processor time (over all processors of Π) that J_i receives from schedule S_I over the interval $[t_1, t_2)$. In other words,*[5]

$$W(S_I, J_i, t_1, t_2) \stackrel{def}{=} \sum_{\pi_k \in \Pi} \left[\int_{t_1}^{t_2} S_I(\pi_k, t, J_i) dt \right]. \tag{1.14}$$

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

$$W_I(S_I, t_1, t_2) \stackrel{def}{=} \sum_{J_i \in I} W(S_I, J_i, t_1, t_2). \tag{1.15}$$

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 1.1 through 1.5, we can now formally define a *valid* schedule S_I with respect to a real-time instance I:

Definition 1.6 (Valid Schedule) *$S_I \in \mathbb{S}_{I,\Pi}$ is valid (with respect to jobs of some real-time instance I and platform Π) 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*

[5]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 (Kolmogorov and Fomin, 1970) and not a Riemann integral.

dissertation, we 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). This assumption excludes certain task systems such as sporadic tasks with relative deadlines greater than the task's period. We are currently working on extending our results to such systems that forbid intra-task parallelism and require that jobs of a task execute in-order.

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* Π *equal to its execution requirement between its release time and deadline).*

1.4.2 Feasibility Analysis

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 τ is *feasible* on processing platform Π 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 Π.

Definition 1.7 (Feasible Instance) *A real-time instance* I *is* feasible *on platform* Π *if and only if there exists* $S_I \in \mathbb{S}_{I,\Pi}$ *such that* S_I *is valid.*

We may extend this definition to recurrent task systems that allow for full migration between processors.

Definition 1.8 (Full-Migration Feasible Task System) *Recurrent task system* τ *in task model* M *is* full-migration feasible *on platform* Π *if and only if for all* $I \in \mathcal{I}^{\mathrm{M}}(\tau)$, I *is a feasible instance on* Π.

For restricted-migration systems (where a job can only execute upon a single processor), we must restrict the definition of feasibility, slightly.

Definition 1.9 (Restricted-Migration Feasible Task System) *Recurrent task system τ in task model* M *is* restricted-migration feasible *on platform* Π *if and only if for all $I \in \mathcal{I}^{\mathrm{M}}(\tau)$, there exists a partition of I into m sets, $I(\pi_1), I(\pi_2), \ldots, I(\pi_m)$, such that for all $\pi_k \in \Pi$, $I(\pi_k)$ is feasible on π_k.*

Finally, for partitioned systems comprised of recurrent tasks, we have the following.

Definition 1.10 (Partition Feasible Task System) *Recurrent task system τ in task model* M *is* partition feasible *on platform* Π *if and only if there exists a partition of τ into m sets, $\tau(\pi_1), \tau(\pi_2), \ldots, \tau(\pi_m)$, such that for all $\pi_k \in \Pi$, $\tau(\pi_k)$ is feasible on π_k.*

A feasibility test labels a task system τ as either "feasible" or "infeasible" on platform Π (*infeasible* is the negation of feasible — i.e., there exists a real-time instance I generated by τ such that there does not exist a schedule for I that is valid for platform Π). An *exact* feasibility analysis test will classify a task system as "feasible" if and only if τ is feasible on Π. A *sufficient* feasibility test may incorrectly classify a feasible task system as "infeasible", but has the property that if τ is classified by the test as "feasible", then τ is in fact feasible on Π.

In a sense, a feasibility test checks whether their exists some algorithm (either online or hypothetical clairvoyant[6]) that will meet all the deadlines of the task system on the processing platform. We will see in Chapter 2 that for completely-specified instances and LL task systems, there are simple, exact feasibility tests for multiprocessor systems. However, for sporadic and more general partially-specified task systems,

[6] A *clairvoyant* scheduling algorithm has knowledge of future job arrivals and therefore does not need to satisfy the restriction of online scheduling algorithms (Definition 1.3).

exact feasibility tests are currently unknown for full- and restricted-migration systems; developing "good" sufficient feasibility test for these general task systems is the focus of Chapter 4.

1.4.3 Schedulability Analysis

For a given scheduling algorithm, task system, and processing platform, schedulability analysis determines whether the given algorithm will *always* meet all deadlines for the task system upon the processing platform.

Definition 1.11 ($\mathcal{A}$-Schedulable) *Recurrent task system τ in task model* M *is* $\mathcal{A}$-schedulable *on platform* Π *if and only if for all $I \in \mathcal{I}^{\mathrm{M}}(\tau)$, $S_I^{\mathcal{A}}$ is a valid schedule on* Π.

Scheduling algorithm $\mathcal{A}$ is an *optimal scheduling algorithm* if, for all τ and Π, τ being feasible on Π implies that τ is also $\mathcal{A}$-schedulable on platform Π. We will see in Chapter 2 that optimal scheduling algorithms exist for full-migration and partitioned scheduling of LL task systems. However, we will show in Chapter 5 that optimal scheduling algorithms cannot exist for sporadic or more general task systems.

Similar to feasibility tests, a schedulability test for algorithm $\mathcal{A}$ labels a task system τ as either "$\mathcal{A}$-schedulable" or "not $\mathcal{A}$-schedulable" on platform Π (a task system τ *not $\mathcal{A}$-schedulable* implies the existence of a real-time instance generated by τ that will miss a deadline when using algorithm $\mathcal{A}$). An *exact* schedulability analysis test will classify a task system as "$\mathcal{A}$-schedulable" if and only if τ is $\mathcal{A}$-schedulable on Π. A *sufficient* schedulability test may incorrectly classify an $\mathcal{A}$-schedulable task system as "not $\mathcal{A}$-schedulable", but has the property that if τ is classified by the test as "$\mathcal{A}$-schedulable", then τ is in fact $\mathcal{A}$-schedulable on Π. Chapters 6 and 7 develop "good" schedulability tests for full-migration and partitioned scheduling algorithms for sporadic and more general task systems.

1.4.4 Evaluating the Effectiveness of a Verification Technique: Resource Augmentation Analysis

In the preceding two subsections (Sections 1.4.2 and 1.4.3), we alluded to "good" verification techniques. For uniprocessor systems, there are both exact feasibility and schedulability tests for all the recurrent task models discussed in Section 1.1 (e.g., see (Baruah, 2003) for discussion of feasibility and schedulability tests for general task systems). Therefore, for uniprocessor systems, it is very clear what constitutes a "good" verification technique; furthermore, there are known optimal scheduling algorithms (such as EDF (Liu and Layland, 1973)). Chapter 5 proves that optimal algorithms for multiprocessor scheduling sporadic and more general task systems cannot exist; Chapter 2 shows that many traditional tests for feasibility and schedulability are not exact. Therefore, it is not entirely clear what constitutes a good verification technique for the multiprocessor scheduling of sporadic and more general task systems. This subsection discusses a possible approach for evaluating real-time verification techniques, and introduce *resource augmentation analysis*, a technique for quantifying "goodness" based on worst-case behavior.

One approach for estimating the effectiveness of a verification technique is via empirical analysis. A typical approach in the real-time systems literature is to randomly generate synthetic task systems (i.e., randomly generate the parameters for tasks) and use the proposed verification technique on the generated task systems. The ratio of randomly generated task systems that are validated by the proposed verification technique (i.e., labeled as either "feasible" or "$\mathcal{A}$-schedulable") to the total number of generated task systems is called the *acceptance ratio*. Such an approach is used by (Park et al., 1995) to determine the effectiveness of a RM schedulability test on LL task systems upon a uniprocessor platform. The acceptance ratio is useful for determining the average-case performance of a scheduling algorithm or verifica-

tion technique over a random distribution of task systems. However, the empirical approach may susceptible to unintended bias (Bini and Buttazzo, 2005). Furthermore, empirical analysis does not give any insight into the worst-case behavior of the verification technique or scheduling algorithm.

A standard way of theoretically evaluating many online algorithms is via a technique known as *competitive analysis*. Let $c(\mathcal{A}, I)$ be the "cost" of the solution produced by algorithm $\mathcal{A}$ over the input I. Let $c_{\text{OPT}}(I)$ be the cost of the solution produced by an optimal algorithm. An algorithm $\mathcal{A}$ has *competitive ratio* of at least $\alpha \geq 1$ if:

$$\max_{I} \frac{c(\mathcal{A}, I)}{c_{\text{OPT}}(I)} \leq \alpha. \tag{1.16}$$

We say that algorithm $\mathcal{A}$ is α-*competitive*. If the goal is to minimize the worst-case cost of an algorithm, then a "good" algorithm will have a small competitive ratio α, while "poor" algorithms will have a large (or unbounded) ratio. The competitive analysis approach is very effective for many optimization problems by identifying good online algorithms.

Unfortunately, the standard competitive analysis approach has serious drawbacks for real-time systems. For a hard real-time system even a single deadline miss may be unacceptable. Therefore, it is difficult to interpret for hard real-time algorithms what the cost of a scheduling algorithm is and what the competitive ratio implies about the algorithm. Even in systems that can tolerate some deadline misses and have define the "cost" of a deadline miss, (Kalyanasundaram and Pruhs, 2000) show that many algorithms that perform well in practice have a large competitive ratio.

To address the shortcomings of standard competitive analysis in real-time scheduling, *resource augmentation* (Phillips et al., 2002) was proposed as a measure of the relative effectiveness and tightness of a given feasibility or schedulability analysis technique. In addition, resource augmentation may be used to indirectly gauge the

effectiveness of a non-optimal scheduling algorithm. Resource augmentation works by comparing a given verification technique against the performance of a hypothetically optimal analysis technique. The resource-augmentation metric of effectiveness used in this dissertation for a given analysis technique is as follows: given any real-time instance that is formally verifiable according to the hypothetical optimal algorithm on the original platform (i.e., feasible on the original platform), our goal is to obtain a constant multiplicative factor by which we must increase the speed of each processor in order for our given analysis test to label the same instance as "feasible" (or "$\mathcal{A}$-schedulable", if using schedulability analysis). That is, we are interested in the minimum speed-up factor necessary to guarantee that our analysis technique verifies that a real-time instance that is optimally schedulable on the original platform is also formally verifiable (according to our sufficient analysis test) on the more powerful, modified platform. We believe that resource-augmentation analysis is particularly useful, as it quantifies the minimum amount of resources that would have to be added to the original platform for a given verification test to validate the same task systems that can be validated by an optimal algorithm; in other words, resource-augmentation quantifies the processing capacity that might be "wasted" in using a non-optimal scheduling algorithm.

Let $\mathcal{V}$ be a verification test (either feasibility or schedulability test). The function $\mathcal{V}$ maps real-time instances and processing platforms to labels (i.e., "feasible" or "infeasible", or "$\mathcal{A}$-schedulable" or "not $\mathcal{A}$-schedulable"). We will now more formally define resource augmentation for $\mathcal{V}$. We will abuse notation slightly and let $s \cdot \Pi$ indicate a platform with m processors where each processor is s times faster ($s \geq 1$) than the processors of Π.

Definition 1.12 (Resource-Augmentation Approximation Ratio for $\mathcal{V}$) *The* resource-augmentation approximation ratio *for verification test $\mathcal{V}$ is the minimum*

s satisfying

$$s \geq \max_{I \in \mathcal{I}:\ I \text{ is feasible on } \Pi} \left\{ \min_{s' \geq 1} \{ s' \mid \mathcal{V}(I, s' \cdot \Pi) \text{ labels } I \text{ “feasible” (“}\mathcal{A}\text{-schedulable”).} \} \right\}. \tag{1.17}$$

Note, that if $\mathcal{V}$ is a schedulability test for $\mathcal{A}$, and if $\mathcal{V}$ has a resource-augmentation approximation ratio of s, we may indirectly say that scheduling algorithm $\mathcal{A}$ has a resource-augmentation approximation ratio of at most s.

1.5 Contributions

The thesis of this dissertation is:

> *Optimal online multiprocessor real-time scheduling algorithms for sporadic and more general task systems are impossible; however, efficient, online scheduling algorithms and associated feasibility and schedulability tests, with provably bounded deviation from any optimal test, exist.*

The above thesis is supported by the following contributions made in this dissertation:

- We describe the relationship between general task models and the well-known workload metrics of *demand-based load* and *maximum job density*. We show that tight upper bounds on demand-based load and maximum job density may be obtained for task systems in each of the models discussed in Section 1.1.
- The best known algorithms for computing demand-based load for sporadic and more general task systems require exponential time in the worst case. We show that for sporadic task systems the demand-based load can be approximated efficiently via a polynomial-time approximation scheme (PTAS).

- We demonstrate the difficulty of the online scheduling of sporadic and more general task systems by proving that optimal online scheduling of these task systems upon multiprocessor platforms is impossible. This results implies that non-optimal online scheduling algorithms for multiprocessor systems are required.

- We derive a restricted-migration feasibility test using demand-based load and maximum job density for real-time instances that has a resource-augmentation approximation ratio of at most $4 - \frac{1}{m}$ where m is the number of processors in platform Π. This feasibility test may be applied to all the partially-specified recurrent task models discussed in Section 1.1.

- We derive a full-migration feasibility test using demand-based load and maximum job density for real-time instances that has a resource-augmentation approximation ratio of at most $\sqrt{2}+1$. Again, this feasibility test may be applied to all the partially-specified recurrent task models discussed in Section 1.1.

- We derive EDF and DM (full-migration) schedulability tests using demand-based load and maximum job density for partially-specified recurrent task systems. The application of this test to DM-scheduled sporadic task systems is discussed and it is shown that the DM-schedulability test has resource-augmentation approximation ratio at most $4 - \frac{1}{m}$; the DM resource-augmentation results is compared with previously known DM-scheduling tests.

- We develop a polynomial-time partitioning algorithm for sporadic task systems when either EDF or DM is used to schedule each individual processor. For this algorithm, we derive schedulability tests for our algorithm using demand-based load and maximum job density, and we show that these tests have a resource-augmentation approximation ratio of at most $4 - \frac{2}{m}$. For a special subset of

<table>
<tr><td rowspan="2">Scheduling:
Paradigm:</td><td colspan="3">Task
Model</td></tr>
<tr><td>Liu & Layland</td><td>Sporadic</td><td>GMF/Recurring/etc..</td></tr>
<tr><td>Uniprocessor
Scheduling</td><td colspan="3">Previous Research
(Well Understood)</td></tr>
<tr><td>Full-Migration
Scheduling</td><td rowspan="4"></td><td colspan="2" rowspan="2">This
Research</td></tr>
<tr><td>Restricted-Migration
Scheduling</td></tr>
<tr><td>Partitioned
Scheduling</td><td rowspan="2"></td><td>Future
Work</td></tr>
<tr><td>Global
Scheduling</td><td></td></tr>
</table>

Table 1.2: The above table shows the contribution of this dissertation where the research space has been categorized by scheduling paradigm versus partially-specified recurrent task model. Observe that most prior work has assumed either uniprocessor scheduling, or multiprocessor scheduling of LL task systems. This dissertation describes contributions to multiprocessor scheduling of sporadic and more general tasks systems. Further research is required for the partitioned scheduling of task systems in models more general than the sporadic task model.

sporadic task systems, known as *constrained-deadline* sporadic task systems, we show that the resource-augmentation approximation ratio is tighter with a ratio of $3 - \frac{1}{m}$.

Table 1.2 places the contributions of this dissertation in the context of previous work. As mentioned in the beginning of the introduction, most prior work on real-time scheduling of recurrent tasks has focused on either uniprocessor scheduling or multiprocessor scheduling of LL task systems. The work contained in this dissertation addresses the multiprocessor feasibility and schedulability of sporadic and more general task systems.

1.6 Organization

This dissertation is organized as follows. In Chapter 2, we survey previous results in multiprocessor real-time scheduling, including the scheduling of arbitrary real-time instances, LL task systems, sporadic task systems, and more general systems. For each of these different models, we present prior results concerning feasibility, schedulability, and optimality. In Chapter 3, we formally introduce the demand-based load and maximum job density characterization of real-time work used throughout this dissertation. We show that both demand-based load and maximum job density metrics may be efficiently computed for any task model discussed in this dissertation. Furthermore, we show that demand-based load may be approximated in polynomial time for sporadic task systems. In Chapter 4, we present our feasibility tests for the full- and restricted-migration scheduling paradigms. In Chapter 5 and Appendix A, we prove that optimal online scheduling of sporadic and more general task systems is impossible. In Chapter 6, we derive schedulability tests for full-migration scheduling of general task systems. In Chapter 7, we develop polynomial-time algorithms for partitioning sporadic task systems and derive schedulability tests for these algorithms. We finally summarize and conclude this dissertation in Chapter 8.

Chapter 2

Related Work: Multiprocessor Real-time Scheduling

As mentioned in the introduction, most prior research in real-time scheduling theory has predominantly focused on uniprocessor systems or the multiprocessor scheduling of the simpler recurrent task systems (namely periodic and LL tasks). In this chapter, we will review some of the prior fundamental results in multiprocessor scheduling of real-time systems. We will begin with the most general characterization of real-time workloads, *arbitrary real-time instances*. For arbitrary real-time instances (sometimes referred to as *independent jobs* in the real-time literature) there have been some fundamental results concerning multiprocessor scheduling. In Section 2.1, we will briefly review these results and discuss how they pertain to the results of this dissertation. We will also identify some shortcomings of these multiprocessor results for arbitrary real-time instances that this dissertation has attempted to address.

After discussing the most general characterization of real-time work, we focus on the strictest partially-specified real-time task model discussed in the introduction: the LL task model. In Section 2.2, we briefly summarize the results that have been obtained for the feasibility and schedulability of task systems in the LL model under

the three paradigms of multiprocessor scheduling (partitioned, full-migration, and restricted-migration) and three paradigms of priority-driven scheduling (fixed-task-priority, fixed-job-priority, and dynamic-priority). We also discuss some fundamental challenges that are present in the multiprocessor scheduling of LL task systems that cause scheduling algorithms that are optimal for uniprocessor scheduling (e.g., EDF) to perform arbitrarily poorly in the multiprocessor setting.

In Section 2.3, we review known results concerning multiprocessor real-time scheduling of sporadic task systems. We first discuss the ineffectiveness of the metrics used in schedulability and feasibility tests for LL task systems on multiprocessors when applied to sporadic task systems. We then give a brief overview of the current state-of-the-art schedulability tests for EDF and DM in the three multiprocessor scheduling paradigms. In Section 2.4, we discuss the small amount of prior work concerning real-time multiprocessor scheduling of task models that generalize the sporadic task model.

2.1 Arbitrary Real-Time Instances

With Section 2.1.1, we begin our overview of related work with a negative result that shows that optimal online scheduling of arbitrary real-time instances is impossible. In Section 2.1.2, we review results that show that despite the impossibility of optimal online scheduling there are real-time multiprocessor scheduling algorithms that have constant factor resource-augmentation approximation ratios in terms of resource-augmentation analysis. Section 2.1.3 describes an essential property for real-time scheduling algorithms known as *predictability*; we briefly discuss its importance. Section 2.1.4 presents a proposed online metric for multiprocessor schedulability of arbitrary real-time instances known as *synthetic utilization*; even though, this result gives a sufficient schedulability test for arbitrary instances, we will show that synthetic

utilization performs arbitrarily poorly in terms of resource augmentation.

2.1.1 Impossibility of Optimal Online Scheduling of Arbitrary Real-Time Instances

In the context of arbitrary real-time instances, a scheduling algorithm $\mathcal{A}$ is optimal if for every $I \in \mathcal{I}$ that is feasible on platform Π, the schedule $S_I^{\mathcal{A}}$ produced by $\mathcal{A}$ on platform Π is valid. EDF is known to be an optimal uniprocessor scheduling algorithm (Liu and Layland, 1973), even for arbitrary real-time instances. Unfortunately, as soon as an additional processor is added to the platform, EDF is no longer optimal; in fact, for arbitrary real-time instance an even stronger negative statement is true:

Theorem 2.1 *For arbitrary real-time instances, no online multiprocessor scheduling algorithm is optimal.*

Variants on this theorem were independently stated and proven by both (Hong and Leung, 1988) and (Dertouzos and Mok, 1989). (Hong and Leung, 1988) gave a slightly stricter result below which immediately implies Theorem 2.1.

Theorem 2.2 (from (Hong and Leung, 1988)) *No optimal on-line multiprocessor scheduling algorithm exists for real-time instances with two or more distinct deadlines (i.e., there exists $J_k, J_\ell \in I$ such that $A_k \neq A_\ell$).*

(Dertouzos and Mok, 1989) show a similar theorem; in addition, their work explored what "knowledge" an optimal online multiprocessor scheduling algorithm required about the real-time instance being scheduled. They proved that even partial information about all the jobs of I (e.g., having knowledge about the execution requirements of instance of all jobs of I but not the arrival times) was not sufficient for the existence of optimal scheduling algorithms for arbitrary real-time instances.

§ **Implications to Online Scheduling of Recurrent Task Systems.** Though the above results may appear to have negative implications on the existence of optimal scheduling algorithms for recurrent task systems, they, in fact, do *not* apply. We will see in the next section (Section 2.2) that optimal online multiprocessor scheduling algorithms do exist for LL task systems. The impossibility results for optimal online scheduling of arbitrary real-time instances do not imply the non-existence of optimal online scheduling algorithms for recurrent task systems due to the difference in the set of feasible instances that must be optimally scheduled in the different contexts. The definition for optimal algorithm $\mathcal{A}$ in the context of arbitrary real-time systems requires that every $I \in \mathcal{I}$ that is feasible on Π be correctly schedule by $\mathcal{A}$; however, the definition of an optimal algorithm $\mathcal{A}'$ for a recurrent task system in model M only requires that if task system τ is feasible on Π, then $\mathcal{A}'$ must correctly schedule every $I' \in \mathcal{I}^{\mathrm{M}}(\tau)$. We have thus restricted the number of feasible instances an optimal algorithm must correctly schedule because $\mathcal{I}^{\mathrm{M}}(\tau) \subset \mathcal{I}$ (see Figure 1.6). For LL task systems, this restriction is sufficient to allow for optimal online scheduling algorithms. However, we will prove in Chapter 5 and Appendix A that the restriction is not sufficient in sporadic and more general task systems, and optimal online scheduling for systems in these models is impossible.

2.1.2 Resource Augmentation Results for Online Scheduling Algorithms

Despite the non-existence of optimal online multiprocessor scheduling algorithms for arbitrary real-time instances, there do exist online scheduling algorithms with constant factor approximation ratios in terms of the resource-augmentation approximation ratio (discussed in Section 1.4.4). (Phillips et al., 2002) studied the behavior of many online multiprocessor scheduling algorithms (both real-time and non-real-time)

when given faster processors than the optimal scheduling algorithm (not necessarily online). In particular, they obtained resource augmentation guarantees for EDF when scheduling arbitrary real-time instances on a multiprocessor platform. The following theorem states the guarantee obtained for EDF:

Theorem 2.3 (from (Phillips et al., 2002)) *If real-time instance $I \in \mathcal{I}$ is feasible on a processing platform comprised of m unit-speed processors, then* EDF *(under the full-migration paradigm) will always meet all deadlines when scheduling I on an m-processor platform where each processor is of speed $2 - \frac{1}{m}$.*

The above result provides a rather strong theoretical guarantee and justification for considering EDF for multiprocessor scheduling. The justification is even more compelling when considering the fact that no online algorithm can have a resource-augmentation approximation ratio better then $1 + \frac{1}{5}$ (also shown in (Phillips et al., 2002)).

§ **Implications to Online Scheduling of Recurrent Task Systems.** The results of Theorem 2.3 have a straightforward implication to the online scheduling of recurrent task systems; specifically, any recurrent task system τ that is feasible on m unit-speed processors is schedulable according EDF on m-processors each of speed $2-\frac{1}{m}$. Of course, since there exist optimal online multiprocessor scheduling algorithms for LL tasks, the lower bound on the resource-augmentation approximation ratio does not directly apply to recurrent tasks.

While Theorem 2.3 is a powerful guarantee on the effectiveness of EDF for scheduling real-time jobs, the theorem (prior to this dissertation) had limited practical applications for the schedulability of sporadic or more general task models. The reason is that prior to the work contained in this dissertation, there did not exist effective non-trivial feasibility tests for sporadic and more general task systems upon multiprocessor platforms; that is, there was no effective method to test the antecedent of Theorem 2.3

for sporadic or more general task systems. The feasibility results contained in this dissertation allow for the real-time system designer to test the antecedent of Theorem 2.3 and thereby directly use Thereom 2.3 to determine the EDF-schedulability of a recurrent task system (Corollary 6.4 in Chapter 6 makes use of this approach).

2.1.3 The Predictability of Multiprocessor Scheduling Algorithms

From a real-time system designer's perspective, the predictability of the system is immensely important. That is, the system should be tolerant of variations in execution provided these variations are within the system's specified parameters. For example, the designer might specify the worst-case execution requirement of each job of a real-time instance I. If the designer verifies that the system is temporally correct when executing instance I under the worst-case executions, a predictable system should continue to be temporally correct under variations in execution where a job executes for less than its worst-case requirement. If the system became unschedulable due to some jobs executing for less than their worst-case execution time, this would represent an *anomaly* in the scheduling algorithm used; for multiprocessor systems, it is not immediately evident that scheduling algorithms are anomaly-free.

(Ha and Liu, 1994) addressed the multiprocessor scheduling of real-time jobs under such variations in execution. In the context and terminology of this dissertation, a system (using scheduling algorithm $\mathcal{A}$) is *predictable* if and only if every $I \in \mathcal{I}$ that is $\mathcal{A}$-schedulable on Π implies that every $I' \in \mathcal{F}(I)$ is also $\mathcal{A}$-schedulable on Π. (Recall that $\mathcal{F}(I)$ is the set of all real-time instances that have the same jobs as I, but smaller or equal execution requirements). (Ha and Liu, 1994) examined the predictability of systems scheduled by *work-conserving* FJP scheduling algorithms. A work-conserving scheduling algorithm always executes a job if it is active and a processor is available

(i.e., processors do not idle while jobs are awaiting execution). Algorithms such as (full-migration) EDF and DM are known to be work-conserving

Theorem 2.4 (from (Ha and Liu, 1994)) *A multiprocessor system scheduled by a work-conserving FJP scheduling algorithm is predictable under preemptions, full-migration, and fixed-job-priority.*

The implications of the above result for multiprocessor systems implies that a predictable, preemptable and full-migration system using a work-conserving FJP algorithm can be validated under the worst-case execution requirements and is immediately guaranteed to be correct under variations in execution that require less than a worst-case amount of time; this observation removes the burden on the designer of validating the system under a potentially large (or infinite) number of executions that may not require the worst-case execution time. Unfortunately, Ha and Liu observed that not all restricted-migration systems are predictable; therefore, this property must be verified for each individual restricted-migration scheduling algorithm.

§ **Implications to Online Scheduling of Recurrent Task Systems.** Theorem 2.4 immediately implies the predictability of preemptable, full-migration systems using work-conserving FJP scheduling algorithms that schedule recurrent task systems. Note that Theorem 2.4 does not represent a necessary condition for a system being predictable. For example, Pfair-based scheduling algorithms (Baruah et al., 1996) are not FJP algorithms or necessarily work-conserving; however, Pfair-based scheduling algorithms are predictable scheduling algorithms for LL task systems.

A concept that is closely related to predictability is *sustainability* (Baruah and Burns, 2006). Informally, a verification test for scheduling algorithm $\mathcal{A}$ (denoted $\mathcal{V}_{\mathcal{A}}$) is sustainable if $\mathcal{V}_{\mathcal{A}}$ determines that recurrent task system τ is $\mathcal{A}$-schedulable on platform Π, then a task system τ' with "reduced" temporal constraints (e.g., each task of τ' is the same except has a larger period — see (Baruah and Burns, 2006)

for more detailed discussion) will also be determined to be $\mathcal{A}$-schedulable by $\mathcal{V}_{\mathcal{A}}$. The advantage of a sustainable verification test is that the designer may validate the system under the "worst-case" temporal constraints, but also be ensured that a system that has reduced temporal constraints will remain verifiable by the same validation technique. It turns out that the schedulability tests presented in Chapters 6 and 7 are sustainable. (We refer the interested reader to (Baruah and Burns, 2006)).

2.1.4 Synthetic Utilization

Thus far, we have only discussed qualitative properties (optimality, predictability, and effectiveness) of multiprocessor scheduling of arbitrary real-time instances. We have not addressed online or *a priori* verification techniques for such instances. An online schedulability test for arbitrary real-time instances has been proposed based on a metric of real-time workload called *synthetic utilization* (Abdelzaher et al., 2004), defined below.

Definition 2.1 (Synthetic Utilization) *The* synthetic utilization, $\mathsf{synth\text{-}util}(I,t)$, *of real-time instance I at time t is:*

$$\mathsf{synth\text{-}util}(I,t) \stackrel{def}{=} \sum_{J_i \in I : A_i \leq t < A_i + D_i} \frac{E_i}{D_i}. \tag{2.1}$$

Table 2.1 presents a brief overview of schedulability tests obtained using synthetic utilization.

While synthetic utilization provides a simple test for the schedulability of a real-time instance, it suffers from a serious drawback: *any verification test using synthetic utilization has an unbounded resource-augmentation approximation ratio.* To see this, consider an instance I comprised of n jobs, all arriving at time-instant 0 and having an execution-requirement of 1, and with the i'th job's deadline at time-instant i. This

SCHEDULING: PARADIGM	SCHEDULING ALGORITHM	
	EDF-based	DM-based
Full-Migration	$\mathsf{synth\text{-}util}(I,t) \leq \frac{m^2}{2m-1}$ (Andersson et al., 2003a)	$\mathsf{synth\text{-}util}(I,t) \leq (3-\sqrt{7})m$ (Lundberg and Lennerstad, 2003)
Restricted-Migration	$\mathsf{synth\text{-}util}(I,t) \leq .31m$ (Andersson et al., 2003b)	?

Table 2.1: A brief overview of the multiprocessor schedulability tests based on synthetic utilization. If a real-time instance I satisfies the test in the above table for all $t \geq 0$, then I is schedulable on m processors according to the algorithm in the header. The reader is referred to the individual papers for greater details on the algorithms considered (essentially EDF or DM with minor modifications).

instance is feasible upon a single unit-capacity processor, yet has synthetic utilization equal to $\sum_{i=1}^{n}(1/i)$, which increases with n. Thus, there exists real-time instances that are feasible on a multiprocessor platform Π (the above instance is feasible even on a uniprocessor), but would require that Π be sped-up arbitrarily fast to verify the schedulability of I. Since the goal of this dissertation is to derive verification tests with constant-factor approximation ratios, we will not consider synthetic utilization further due to its non-optimal performance in the worst case.

2.2 LL Tasks

While the previous section shows that there are scheduling algorithms for arbitrary real-time instances with constant-factor approximation ratios, previous work has not developed *a priori* verification tests with such approximation ratios. A fundamental reason for the lack of good verification tests is the set of different real-time instances that a verification test must implicitly consider consists of the entire space of real-time instances $\mathcal{I}$. In simple terms, the verification technique must handle an enormous set of possibilities with very little information on what real-time instance will actually be

generated at run-time. On the other hand, a partially-specified real-time task system τ (such as a task system in the LL model) generates a significantly smaller set of possible real-time instances; furthermore, the types of jobs that a task system generates are finite. For a partially-specified task system, there is also partial knowledge about the characteristic of the jobs generated. Therefore, it is not surprising that there exist exact feasibility tests, and either near-optimal or optimal schedulability tests for LL task systems.

In this section, we describe prior work on the multiprocessor scheduling of LL task systems. Section 2.2.1 introduces and discusses *system utilization*, a metric used in feasibility and schedulability test. Section 2.2.2 discusses a challenge in the online scheduling of LL task systems. Section 2.2.3 gives an overview of the schedulability test obtained for various scheduling algorithms for LL task systems.

2.2.1 Task Utilization

An effective characterization of the real-time workload produced by a LL task is called *system utilization*:

$$\mathsf{system\text{-}util}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \frac{e_i}{p_i}. \tag{2.2}$$

The *task utilization* of task τ_i is denoted by $u_i \stackrel{\text{def}}{=} \frac{e_i}{p_i}$. Informally, task utilization represents the fraction of computational capacity that a task requires on a single processor. The amount of execution over any interval of length t on processing platform Π that a task τ_i requires is upper bounded by $u_i \times t$. (Horn, 1974) pointed out that, system utilization may be used as an exact test for the feasibility of a LL task system.

Theorem 2.5 *A LL task system τ is feasible on an m-processor platform Π if and*

only if

$$\mathsf{system\text{-}util}(\tau) \leq m. \tag{2.3}$$

Throughout this dissertation, the following task parameter representing the maximum utilization of any task in τ will also be useful.

$$\mathsf{max\text{-}util}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau}\{u_i\}. \tag{2.4}$$

2.2.2 Dhall's Effect

The test of Theorem 2.5 is a simple, exact feasibility test for LL task systems on multiprocessor platforms. For LL task systems scheduled on a uniprocessor platform, the test $\mathsf{system\text{-}util}(\tau) \leq 1$ is an exact schedulability test for EDF. The next example shows that, unfortunately, a simple, exact utilization-based test is not possible for EDF-schedulability on multiprocessor platforms.

Example 2.1 Consider platform Π with m processors, and LL task system τ comprised of $m+1$ tasks. Let $\tau_1 \stackrel{\text{def}}{=} (\epsilon, 1), \tau_2 \stackrel{\text{def}}{=} (\epsilon, 1), \ldots, \tau_m \stackrel{\text{def}}{=} (\epsilon, 1)$ and $\tau_{m+1} \stackrel{\text{def}}{=} (1.1, 1.1)$. (Recall that a LL task τ_i is specified by the pair (e_i, p_i)). By Theorem 2.5, τ is feasible on Π. It is easy to see that if the system is scheduled by EDF on Π and all tasks generate a job simultaneously at time $t = 0$, a job of task τ_{m+1} will miss its deadline for all $\epsilon > 0$ at time $t = 1.1$ (see Figure 2.1 for a visual depiction). Notice that $\mathsf{system\text{-}util}(\tau) = 1 + m\epsilon$, which implies $\lim_{\epsilon \to 0} \mathsf{system\text{-}util}(\tau) = 1$. ■

Therefore, there exist LL task systems with utilization approaching one that are not EDF-schedulable on a multiprocessor platform comprised of m processors. This effect of task systems having low utilization but being unschedulable on a multiprocessor system is known as *Dhall's effect* (Dhall and Liu, 1978) ((Andersson and Jonsson,

2000) first coined the term). The effect may be observed for other online algorithms such as RM.

2.2.3 Overview of Schedulability Tests

Over the past twenty-five years, researchers have made significant progress in addressing the challenges of multiprocessor scheduling of LL task systems and have developed techniques to overcome Dhall's effect. In Table 2.2, we will list the schedulability tests for each multiprocessor scheduling paradigm and priority-driven algorithm class. We refer the reader to the references listed for each result for further details. As evident

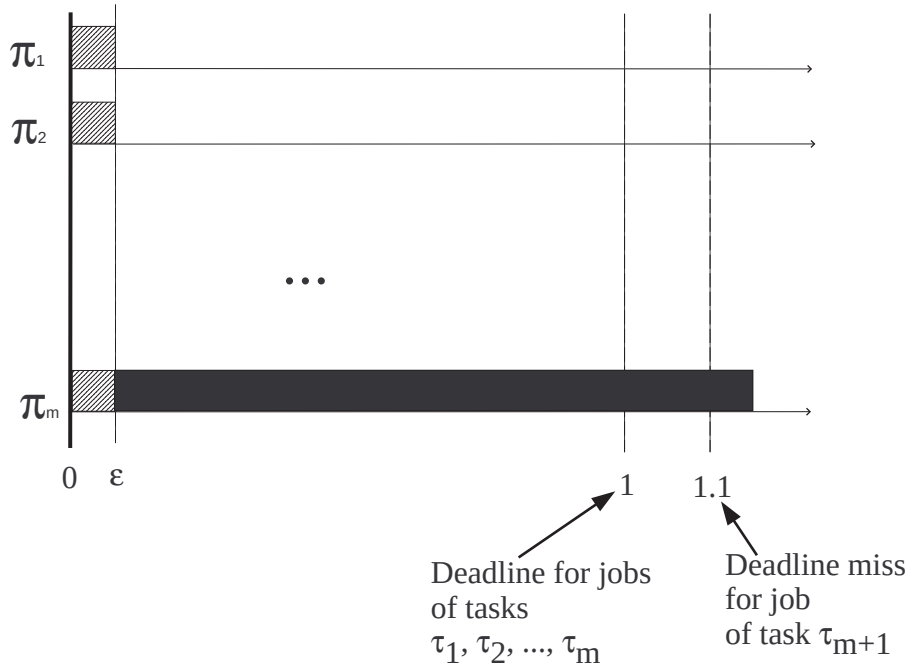


Figure 2.1: Illustration of Dhall's effect for task system in Example 2.1. The lightly shaded jobs correspond to jobs of $\tau_1, \ldots \tau_m$ arriving at time zero with $(e_i, p_i) = (\epsilon, 1)$. The black job corresponds to the job of $\tau_{m+1} = (1.1, 1.1)$ also arriving at time zero. Since the jobs of $\tau_1, \ldots \tau_m$ have deadline at time $t = 1$, they will execute (according to EDF on all m processors in the interval from $[0, \epsilon)$. Thus, the job of τ_{m+1} cannot execute until $t = \epsilon$ and complete at $t = 1.1 + \epsilon$, missing its deadline at time $t = 1.1$. However, (Devi, 2006) shows that the maximum amount of time by which any such job misses its deadline (called the *tardiness* of the job) is bounded.

<table>
<tr><td>Scheduling:
Paradigm</td><td colspan="3">Priority-Driven Class</td></tr>
<tr><td></td><td>FTP</td><td>FJP</td><td>DP</td></tr>
<tr><td>Full-
Migration</td><td>$U \leq \frac{m}{2}(1-\alpha)+\alpha$, if $\alpha < \frac{1}{3}$
$U \leq \frac{m+1}{3}$, otherwise
(Bertogna et al., 2005b)</td><td>$U \leq m(1-\alpha)+\alpha$, if $\alpha < \frac{1}{2}$
$U \leq \frac{m+1}{2}$, otherwise
(Srinivasan and Baruah, 2002)</td><td>$U \leq m$
(Horn, 1974)</td></tr>
<tr><td>Restricted-
Migration</td><td rowspan="2">$U \leq \frac{m+1}{2}$
(Andersson and Jonsson, 2003)</td><td colspan="2">$U \leq m(1-\alpha)+\alpha$, if $\alpha < \frac{1}{2}$
$U \leq \frac{m+1}{2}$, otherwise
(Baruah and Carpenter, 2003)</td></tr>
<tr><td>Partitioned</td><td colspan="2">$U \leq \frac{\beta m+1}{\beta+1}$
(Lopez et al., 2004)</td></tr>
</table>

Table 2.2: The known multiprocessor schedulability tests for LL task systems. Columns represent the priority-driven class of the scheduling algorithm; the rows represent the multiprocessor scheduling paradigm. To save space in the table, we let U denote $\mathsf{system\text{-}util}(\tau)$, α denote $\mathsf{max\text{-}util}(\tau)$, and β denote $\lfloor \frac{1}{\mathsf{max\text{-}util}(\tau)} \rfloor$. If a task system τ satisfies the test in an entry, then τ is schedulable on m processors according to some algorithm in the entry's associated priority class and multiprocessor paradigm. This table is similar to the ones presented in (Carpenter et al., 2003; Devi, 2006); however, the entries have been updated to reflect newer results.

by the existence of optimal scheduling algorithms, exact feasibility tests, and effective schedulability tests for each different class of algorithm, we may observe that the multiprocessor scheduling of LL task systems is fairly well understood.

2.3 Sporadic Tasks

Similar to LL task systems, there are known exact uniprocessor feasibility tests for sporadic task systems (Baruah et al., 1990a). Unfortunately, for the multiprocessor scheduling of sporadic task systems there are currently no known exact feasibility tests. In fact, we will see that prior to the work in this dissertation, the only non-trivial work in the multiprocessor scheduling of sporadic task systems considered full-migration FJP and FTP scheduling algorithms.

In Section 2.3.1, we discuss the shortcomings of traditional workload metrics for the feasibility of sporadic task systems. In Section 2.3.2, we discuss related work on the partitioned scheduling of sporadic task systems. In Section 2.3.3, we review some recent results in the full-migration scheduling of sporadic tasks systems. To the best of our knowledge, there does not exist any significant specific work on restricted-migration scheduling of sporadic task systems.

2.3.1 Limitation of Traditional Workload Metrics

For sporadic task τ_i (where it is possible that d_i is *not* equal to p_i), $u_i \times t$ is no longer an upper bound on the execution demand of τ_i over any interval of length t. It is easy to see that $\mathsf{system\text{-}util}(\tau) \leq m$ is a necessary condition for a sporadic task system being feasible upon an m-processor platform. However, this condition is not sufficient for sporadic task system feasibility. In fact, it can be shown that there exist infeasible task systems with arbitrarily small utilization. This is illustrated in the following example:

Example 2.2 Consider the following sporadic task system consisting of three tasks to be scheduled on a multiprocessor system comprised of two unit-capacity processors:

$$\tau = \{\tau_1 = (1, 1, r), \tau_2 = (1, 1, r), \tau_3 = (1, 1, r)\}.$$

where $r \geq 2$. Observe that $\mathsf{system\text{-}util}(\tau) = 3/r \leq 3/2 \leq 2$, and $\lim_{r\to\infty} \mathsf{system\text{-}util}(\tau) = 0$; however, if each task of τ releases a job at time 0, each job must complete one unit of execution by time 1. There is no way to schedule τ over the interval $[0, 1)$; therefore, τ is infeasible on two processors. ■

An upper bound on execution demand for sporadic task systems over any interval with arbitrary relative deadlines is:

$$\mathsf{system\text{-}density}(\tau) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \mathsf{task\text{-}density}(\tau_i), \tag{2.5}$$

where

$$\mathsf{task\text{-}density}(\tau_i) = \frac{e_i}{\min(d_i, p_i)}. \tag{2.6}$$

Another useful workload metric for sporadic task systems is

$$\mathsf{max\text{-}task\text{-}density}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} \mathsf{task\text{-}density}(\tau_i). \tag{2.7}$$

(The quantity $\mathsf{task\text{-}density}(\tau_i)$ is referred to as the *density* of τ_i.) It was shown in (Ghazalie and Baker, 1995) that $\mathsf{system\text{-}density}(\tau) \leq 1$ is a sufficient condition for the preemptive uniprocessor feasibility of sporadic task systems — this result is easily extended to show that $\mathsf{system\text{-}density}(\tau) \leq m$ is a sufficient condition for feasibility upon a multiprocessor platform comprised of m unit-capacity processors. However, this condition is not necessary for feasibility: consider the following example.

Example 2.3 Given a sporadic task system τ consisting of n tasks to be scheduled on a single preemptive processor. The i'th task has execution-requirement 1, relative deadline i, and inter-arrival separation n. It may be verified (see, e.g., (Baruah et al., 1990b)) that this system is feasible. Its density $\mathsf{system\text{-}density}(\tau) = \sum_{i=1}^{n}(1/i)$, which grows without bound with increasing n. This example illustrates that there are sporadic task systems τ of arbitrarily high density, $\mathsf{system\text{-}density}(\tau)$, which are feasible. ■

In summary, with respect to the preemptive scheduling of a sporadic task system τ on a multiprocessor platform comprised of m unit-capacity processors:

1. $\mathsf{system\text{-}util}(\tau) \times t$ is a lower bound on execution demand over any interval of length t in any real-time instance where task generate jobs as soon as legally

possible: $\mathsf{system\text{-}util}(\tau) \leq m$ is a necessary condition for feasibility;

2. $\mathsf{system\text{-}density}(\tau) \times t$ is an upper bound on execution demand over any interval of length t in any real-time instance where task generate jobs as soon as legally possible: $\mathsf{system\text{-}density}(\tau) \leq m$ is a sufficient condition for feasibility;

3. when $d_i = p_i$ for $\tau_i \in \tau$ (i.e., a LL task system), the two coincide, giving us a necessary and sufficient condition;

4. when $d_i \neq p_i$ the two bounds may leave a gap, where there is uncertainty whether a task set is feasible; Examples 2.2 and 2.3 show the gap may be large.

The conceptual relationship of $\mathsf{system\text{-}util}(\tau)$ and $\mathsf{system\text{-}density}(\tau)$ is illustrated in Figure 2.2. It can be seen that there is a region of uncertainty between the lower and upper bounds. In the next chapter (Chapter 3) we seek to increase the precision of determining the feasibility (or infeasibility) of sporadic task systems by considering a more accurate workload metric.

2.3.2 Partitioned Scheduling

For LL systems, partitioned feasibility-analysis can be transformed to a bin-packing problem (Johnson, 1973) and shown to be NP-hard in the strong sense; sufficient feasibility tests for various bin-packing heuristics have recently been obtained (Lopez et al., 2000; Lopez et al., 2004) (as shown in Table 2.2). For sporadic task systems, the intractability result continues to hold. However, the bin-packing heuristics and related analysis of (Lopez et al., 2000; Lopez et al., 2004) do not trivially extend. To our knowledge, there have been no prior non-trivial positive theoretical results concerning partitioned feasibility analysis of constrained and arbitrary sporadic task systems — "trivial" results include the obvious ones that τ is feasible on m processors if **(i)** it is feasible on a single processor; or **(ii)** the system obtained by replacing each task τ_i by a

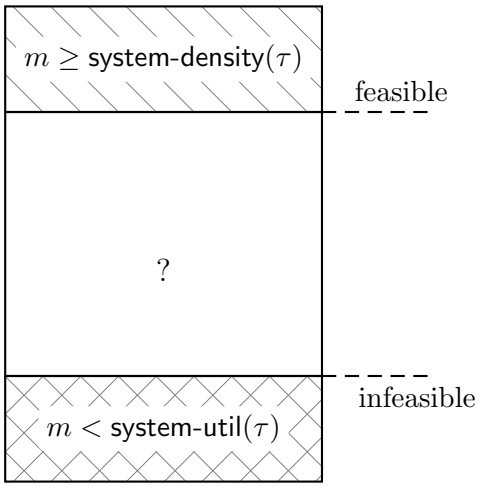


Figure 2.2: The entire rectangle represents the space of all possible sporadic task systems (i.e., $\mathcal{I}^{\mathrm{S}}$). The region shaded by the slanted lines on top represents the space of task systems feasible on m processors according to the system-density(τ)-based test. The crosshatched region at the bottom represents the space of task systems deemed infeasible according to the system-util(τ)-based test. The unshaded region with the question mark is referred to as the *region of uncertainty* because neither of the tests can determine the feasibility of the task systems. This dissertation seeks to narrow the region of uncertainty for multiprocessor systems.

task $\tau_i' = (e_i, \min(d_i, p_i), \min(d_i, p_i))$ is deemed feasible using the heuristics presented in (Lopez et al., 2000; Lopez et al., 2004). There has been some empirical work on partitioning sporadic task systems; Sáez, et al. (Saez et al., 1998) describe and experimentally evaluate partitioning heuristics when the constraint that deadlines equal periods is removed. In Chapter 7, we address the partitioned scheduling of sporadic task systems by developing partitioning algorithms with a constant-factor approximation ratio.

2.3.3 Full-Migration Schedulability Tests

As mentioned at the beginning of this section, there are no known non-trivial feasibility tests for multiprocessor sporadic systems prior to the results of this dissertation that are not associated with a schedulability test. Fortunately, there does exist some

work on schedulability tests. In this subsection, we briefly present some of the known full-migration schedulability tests for the FTP and FJP scheduling of sporadic task systems. A much more thorough overview can be found in (Baker and Baruah, 2007). Though the tests presented have been shown to be reasonably effective empirically, there are no known resource-augmentation approximation ratios associated with any of the tests presented in this section. Chapter 6 will derive schedulability tests for both FTP and FJP scheduling with constant-factor resource-augmentation approximation ratios; in addition, a brief theoretical comparison of FTP schedulability tests for sporadic task systems will be made. We now present the prior schedulability tests.

§ **FTP Scheduling.** The first test presented in this section was developed in (Baker and Cirinei, 2006) and is valid for sporadic task systems where tasks are indexed by decreasing priority:

Theorem 2.6 (from (Baker and Cirinei, 2006)) *Let τ be a sporadic task system $\{\tau_1, \ldots, \tau_n\}$ ordered by decreasing priority. A task $\tau_k \in \tau$ is schedulable on m unit-capacity processors according to static-priority scheduling if there exists $\lambda \in \{\mathsf{task\text{-}density}(\tau_k)\} \cup \{u_i | u_i \geq \mathsf{task\text{-}density}(\tau_k) \wedge \tau_i \in \tau\}$ such that either*

$$\sum_{i<k} \min\left(\beta_k(i,\lambda), 1-\lambda\right) < m(1-\lambda) \tag{2.8}$$

or,

$$\left[\sum_{i<k} \min\left(\beta_k(i,\lambda), 1-\lambda\right) = m(1-\lambda)\right] \bigwedge \left[\exists i \neq k : 0 < \beta_k(i,\lambda) < 1 - \mathsf{task\text{-}density}(\tau_k)\right] \tag{2.9}$$

where

$$\beta_k(i,\lambda) \stackrel{def}{=} \begin{cases} u_i \left(1 + \max\left(0, \frac{p_i - e_i}{d_k}\right)\right) , & \text{if } u_i \leq \lambda \\ u_i \left(1 + \max\left(0, \frac{d_i + p_i - e_i - \lambda d_i/u_i}{d_k}\right)\right) , & \text{otherwise.} \end{cases} \tag{2.10}$$

If we restrict our attention to sporadic task systems where each task $\tau_i \in \tau$ has $d_i \leq p_i$, (Bertogna et al., 2005a) derived the following test:

Theorem 2.7 (from (Bertogna et al., 2005a)) *Let τ be a sporadic task system $\{\tau_1, \ldots, \tau_n\}$ ordered by decreasing priority with $d_i \leq p_i$ for each task τ_i. A task $\tau_k \in \tau$ is schedulable on m unit-capacity processors according to static-priority scheduling if*

$$\sum_{i=1}^{k-1} \min(\beta_k(i), 1 - \frac{e_k}{d_k}) < m(1 - \frac{e_k}{d_k}) \tag{2.11}$$

or,

$$\sum_{i=1}^{k-1} \min(\beta_k(i), 1 - \frac{e_k}{d_k}) = m(1 - \frac{e_k}{d_k}), \quad \left(\text{if } \exists i \neq k : \beta_k(i) \leq 1 - \frac{e_k}{d_k}\right) \tag{2.12}$$

where

$$\beta_k(i) \stackrel{def}{=} \frac{e_i N_k(i) + \min\left(e_i, \max\left(0, d_k - p_i N_k(i) + d_i - e_i\right)\right)}{d_k} \tag{2.13}$$

and

$$N_k(i) \stackrel{def}{=} \left(\left\lfloor \frac{d_k - e_i}{p_i} \right\rfloor + 1 \right). \tag{2.14}$$

§ **FJP Scheduling.** Research has been done on the full-migration EDF-scheduling of sporadic task systems.

Theorem 2.8 (from (Baker, 2005a)) *Let τ be a sporadic task system $\{\tau_1, \ldots, \tau_n\}$. A task $\tau_k \in \tau$ is schedulable on m unit-capacity processors according to* EDF *if there exists $\lambda \in \{\mathsf{task\text{-}density}(\tau_k)\} \cup \{u_i | u_i \geq \mathsf{task\text{-}density}(\tau_k) \wedge \tau_i \in \tau\}$ such that*

$$\sum_{\tau_i \in \tau} \min\left(\beta_k(i,\lambda), 1\right) \leq m(1-\lambda) + \lambda \tag{2.15}$$

where

$$\beta_k(i,\lambda) \stackrel{def}{=} \begin{cases} u_i \left(1 + \max\left(0, \frac{p_i - e_i}{d_k}\right)\right) \ , & \text{if } u_i \leq \lambda \\ u_i \left(1 + \frac{p_i}{d_k}\right) - \lambda \frac{d_i}{d_k} \ , & \text{if } u_i > \lambda \wedge d_i \leq p_i \\ u_i \left(1 + \frac{p_i}{d_k}\right) \ , & \text{otherwise} \end{cases} \tag{2.16}$$

Again, if we restrict our attention to sporadic task systems where each task $\tau_i \in \tau$ has $d_i \leq p_i$, (Bertogna et al., 2005b) derived the following test:

Theorem 2.9 (from (Bertogna et al., 2005b)) *Let τ be a sporadic task system $\{\tau_1, \ldots, \tau_n\}$ ordered by decreasing priority with $d_i \leq p_i$ for each task τ_i. A task $\tau_k \in \tau$ is schedulable on m unit-capacity processors according to static-priority scheduling if*

$$\sum_{i=1}^{k-1} \min(\beta_k(i), 1 - \frac{e_k}{d_k}) < m(1 - \frac{e_k}{d_k}) \tag{2.17}$$

or,

$$\sum_{i=1}^{k-1} \min(\beta_k(i), 1 - \frac{e_k}{d_k}) = m(1 - \frac{e_k}{d_k}), \quad \left(\text{if } \exists i \neq k : \beta_k(i) \leq 1 - \frac{e_k}{d_k}\right) \tag{2.18}$$

where

$$\beta_k(i) \stackrel{def}{=} \frac{e_i N_k(i) + \min\left(e_i, \max\left(0, d_k - p_i N_k(i)/e_i\right)\right)}{d_k} \tag{2.19}$$

and

$$N_k(i) \stackrel{def}{=} \max\left(0, \left(\left\lfloor \frac{e_k - e_i}{p_i} \right\rfloor + 1\right)\right). \tag{2.20}$$

In this subsection, we have presented two tests each for both FJP and FTP scheduling. Unfortunately, it turns out that these tests are incomparable, in the sense that there exist sporadic task systems that are schedulable according to one test, but not the other. Therefore, determining the efficacy of each test is a non-trivial challenge. We refer the interested reader to (Baker, 2006b) for an empirical comparison of these various tests.

2.4 More General Task Models

Research on the multiprocessor scheduling of task systems that are more general than the sporadic task model is virtually nonexistent. The only reference we are aware of is on the distributed scheduling of *distributed generalized multiframe* (DGMF) tasks (Chen et al., 2000). This model is identical to the GMF model presented in Section 1.1.2.3, except a vector $\vec{h_i} \stackrel{\text{def}}{=} \{h_1, h_2, \ldots, h_{N_i}\}$ is added to the task specification. The value $h_k \in \{h_1, h_2, \ldots, h_{N_i}\}$ indicates which host (i.e., processor) the k'th job in the GMF sequence will execute upon; that is, the task specification indicates the job assignment to the processor. Tests are presented in (Chen et al., 2000) for analyzing the FJP schedulability of any DGMF task using uniprocessor schedulability analysis techniques. However, we consider this task model to lie beyond the scope of this dissertation, due to the fact that after task specification the system does not require any decisions about job-processor assignment, essentially sidestepping many of the multiprocessor challenges by statically assigning jobs to processors. Therefore, we will not consider this model further, but focus on other general models where the processor assignment is not included in the task specification. Chapters 4 and 6 will present feasibility and full-migration schedulability tests for general recurrent task systems.

2.5 Summary

In this chapter, we reviewed some of the prior fundamental research in real-time multiprocessor scheduling. For arbitrary real-time instances (Section 2.1), we saw that while optimal online scheduling is impossible, scheduling algorithms with constant-factor approximation ratios exist; however, there are currently no known verification techniques with such approximation ratios for arbitrary real-time instances. The LL task model (Section 2.2) allows for optimal scheduling algorithms, despite many online algorithms suffering from Dhall's effect. Scheduling algorithms have been proposed with effective schedulability tests (see Table 2.2). For the slightly more general sporadic task model (Section 2.3), only recently have researchers focused on full-migration schedulability tests; unfortunately, these tests currently have no known resource-augmentation guarantee. Feasibility and schedulability tests for more general task systems (Section 2.4) on multiprocessor platforms has been a completely open question.

The remaining chapters of this dissertation will address some of the limitations mentioned in this section. Primarily, we will develop feasibility and schedulability tests with constant-factor resource-augmentation approximation ratios for sporadic and more general task systems.

Chapter 3

A Metric of Real-time Workload: Demand-Based Load and Maximum Job Density

The previous chapter highlighted the shortcomings of the standard real-time workload metrics used in verification tests for multiprocessor feasibility and schedulability of sporadic and more general task systems. In this chapter, we suggest using a different well-known characterization of real-time workload: the combination of *demand-based load* (referred to as just *load*, interchangeably) and *maximum job density*. In this chapter we show that these two workload characterizations are closely related to feasibility on multiprocessor platforms. Additionally, we can derive values for both demand-based load and maximum job density from large classes of partially-specified recurrent task systems.

The remainder of this chapter is organized as follows. In Section 3.1, we formally define the concepts of *demand*, *maximum job density*, and *demand-based load*. In Section 3.2, we show that maximum job density and demand-based load can be used in a necessary conditions for the feasibility of a real-time instance upon a multiprocessor

platform. In Section 3.3, we describe the relationship between the load and maximum job density metric for real-time instances and the real-time instances produced by recurrent task systems; it is shown that for many recurrent task models, tight upper bounds on maximum job density and demand-based load may be obtained on the real-time instances that are generated by task systems in these models. In Section 3.4, we describe both exact algorithms and a *polynomial-time approximation scheme* (PTAS) for calculating demand-based load for a sporadic task system.

3.1 Definitions

It is useful, for the purpose of formal analysis, to quantify the amount of computation required over an interval by a real-time instance. We call this quantity the *demand* over the interval. Informally, demand is an indication of how "temporally constrained" the system is over that interval. Below is a more formal definition of demand.

Definition 3.1 (Demand of a Real-Time Instance I) *The* demand *of a real-time instance over a time interval $[t_1, t_2]$ is the sum of the execution requirements of all jobs in the instance that have both their arrival times and their deadlines within the interval:*

$$\mathsf{demand}(I, t_1, t_2) \stackrel{def}{=} \sum_{(J_i \in I) \wedge (t_1 \leq A_i) \wedge (A_i + D_i \leq t_2)} E_i. \tag{3.1}$$

Another useful indicator of how temporally constrained the system might be is the maximum ratio of a job's execution time to its scheduling window. If this ratio, called *maximum job density*, is high then jobs of a real-time instance may require a large fraction of processing time on the platform. Below is the formal definition of maximum job density.

Definition 3.2 (Maximum Job Density of Real-Time Instance I) *The maximum job density of real-time instance I is the maximum ratio of any job's execution requirement to its relative deadline:*

$$\mathsf{max\text{-}job\text{-}density}(I) \stackrel{def}{=} \max_{J_i \in I}(E_i/D_i). \tag{3.2}$$

Intuitively, $\mathsf{max\text{-}job\text{-}density}(I)$ represents the maximum computational demand of any *individual* job.

The *demand-based load* of a real-time instance I represents the maximum *cumulative* computational demand of any subset of the set of jobs, in I. Informally, the load may be interpreted as a lower bound on the minimum number of processors that real-time instance I would require to meet the deadlines of jobs over all intervals. More formally,

Definition 3.3 (Demand-Based Load of Real-Time Instance I) *The demand-based load of real-time instance I, $\mathsf{load}(I)$, is the maximum ratio, over all positive intervals, of the demand of I over the interval to the interval length:*

$$\mathsf{load}(I) \stackrel{def}{=} \max_{t_1 < t_2} \frac{\mathsf{demand}(I, t_1, t_2)}{t_2 - t_1}. \tag{3.3}$$

3.2 Infeasibility Test

The parameters load and max-job-density turn out to be very closely related to the feasibility of a real-time instance on a multiprocessor platform; in fact, these two parameter may be used in a necessary condition for feasibility, as the lemma below shows:

Lemma 3.1 *If real-time instance I is feasible on an identical multiprocessor platform comprised of m unit-capacity processors, then*

$$\mathsf{max\text{-}job\text{-}density}(I) \leq 1 \quad \text{and} \quad \mathsf{load}(I) \leq m.$$

Proof: The first condition follows from the observation that on a unit-capacity processor, a job that meets its deadline by executing continually between its arrival time and its deadline has a maximum job density of one; hence, one is an upper bound on the density of any job. Taken over all jobs in I, this observation yields the first condition.

For the second condition, the requirement that $\mathsf{load}(I) \leq m$ is obtained by considering a set of jobs of I that defines $\mathsf{load}(I)$; i.e., the jobs over an interval $[t_1, t_2)$ such that all jobs arriving in, and having deadlines within, this interval have a cumulative execution requirement equal to $\mathsf{load}(I) \times (t_2 - t_1)$. The total amount of execution that all these jobs may receive over $[t_1, t_2)$ is equal to $m \times (t_2 - t_1)$; hence, $\mathsf{load}(I) \leq m$. ■

If the converse of Lemma 3.1 were to hold, we would have an exact necessary and sufficient condition for migratory feasibility analysis. Unfortunately, the converse of Lemma 3.1 does *not* hold, as is illustrated by the following example.

Example 3.1 Consider the real-time instance I consisting of the three jobs J_1, J_2, and J_3. All three jobs arrive at time 0; jobs J_1 and J_2 have execution requirement of one and a deadline at time 1; and J_3 has an execution requirement of two and a deadline at time 2.

$\mathsf{max\text{-}job\text{-}density}(I)$ equals $\max\{1/1, 1/1, 2/2\}$, which is equal to 1.

Since all arrival times and deadlines are at time-instants $0, 1$, and 2, $\mathsf{load}(I)$ can

be computed by considering the intervals $[0, 1)$, $[0, 2)$, and $[1, 2)$:

$$\mathsf{load}(I) = \max\left(\frac{1+1}{1}, \frac{1+1+2}{2}, \frac{0}{1}\right) = 2 \ .$$

Thus, instance I satisfies the conditions of Lemma 3.1 for $m = 2$; however, it is easy to see that all three jobs cannot be scheduled to meet their deadlines on two unit-capacity processors (since $m = 2$).

■

Lemma 3.1 and Example 3.1 tell us that, while every instance I that is feasible upon a platform comprised of m unit-capacity processors has $\mathsf{load}(I) \leq m$ and $\mathsf{max\text{-}job\text{-}density}(I) \leq 1$, not every instance I' with $\mathsf{load}(I') \leq m$ and $\mathsf{max\text{-}job\text{-}density}(I') \leq 1$ is feasible on such a platform. Chapters 4, 6, and 7 will further explore multiprocessor feasibility and schedulability tests based on load and max-job-density.

3.3 Demand-Based Load of Partially-Specified Recurrent Task Systems

The infeasibility test of the previous section and the feasibility and schedulability tests described in the remainder of this dissertation require that load parameter $\mathsf{load}(I)$ and the job density parameter $\mathsf{max\text{-}job\text{-}density}(I)$ of the real-time instance I being analyzed be known. Thus, it may seem to the reader that the workload characterizations of $\mathsf{load}(I)$ and $\mathsf{max\text{-}job\text{-}density}(I)$ are of limited interest for infinite real-time instances — especially since we had argued in the introduction, many real-time systems are comprised of collections of independent recurrent real-time tasks, each of which generates a potentially infinite sequence of jobs. However, in this section we will show

how, given the specifications of such a real-time system, we may determine bounds on the load and max-job-density parameters of any real-time instance that could be generated by the system during run-time. Our approach is based upon the concept of the *demand-bound function* (DBF) of recurrent real-time tasks; in Section 3.3.1 below, we describe this concept and explain how it relates to determining load and max-job-density. In Section 3.3.1, we briefly discuss the computational complexity of the DBF approach towards multiprocessor feasibility and schedulability of general task models.

3.3.1 The DBF Abstraction

We start out with a definition of the demand-bound function.

Definition 3.4 (Demand-Bound Function) *Let τ_i denote a recurrent real-time task, and t a non-negative real number. The demand-bound function* $\text{DBF}(\tau_i, t)$ *denotes the maximum cumulative execution requirement that could be generated by jobs of τ_i that have both ready times and deadlines within any time interval of duration t.*

The demand-bound function is efficiently determined for all the recurring real-time task models mentioned in this dissertation; algorithms for doing so for the LL and sporadic task models are to be found in (Baruah et al., 1990b), for the multiframe and generalized multiframe models in (Baruah et al., 1999), and for the DAG-based model in (Baruah, 2003). As an illustrative example, we show the formula for computing DBF for a task specified according to the sporadic task model — a formal proof of the formula may be found in (Baruah et al., 1990b). Recall from the introduction that a sporadic task τ_i be represented by the three-tuple (e_i, d_i, p_i), with the interpretation that this task generates an infinite sequence of jobs each with execution requirement e_i and relative deadline d_i, and with the arrival times of successive jobs separated by at least p_i time units. For such a task, it has been shown (Baruah et al., 1990b)

that the cumulative execution requirement of jobs of τ_i over an interval $[t_o, t_o + t)$ is maximized if one job arrives at the start of the interval — i.e., at time-instant t_o — and subsequent jobs arrive as rapidly as permitted — i.e., at instants $t_o + p_i$, $t_o + 2p_i$, $t_o + 3p_i, \ldots$ (such a sequence where the jobs of a task arrive as soon as legally allowable is known as the *synchronous arrival sequence*); Equation (3.4) below follows directly (Baruah et al., 1990b):

$$\text{DBF}(\tau_i, t) \stackrel{\text{def}}{=} \max\left(0, \left(\left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1\right) \times e_i\right). \tag{3.4}$$

Given that we know how to determine the DBF for many of the important recurrent real-time task models, we now discuss how to compute the value of $\mathsf{load}(I)$ for any real-time instance I that is generated by a collection of recurrent real-time tasks $\tau = \{\tau_1, \tau_2, \ldots\}$. The maximum cumulative execution requirement by jobs in I over any time interval $[t_1, t_2)$ is bounded from above by the sum of the maximum execution requirements of the individual tasks in τ:

$$\mathsf{demand}(I, t_1, t_2) \leq \left(\sum_{\tau_i \in \tau} \text{DBF}(\tau_i, t_2 - t_1)\right).$$

From the definition of load (Equation 3.3), it follows that

$$\mathsf{load}(I) \leq \max_{t \geq 0} \left\{ \frac{\sum_{\tau_i \in \tau} \text{DBF}(\tau_i, t)}{t} \right\}. \tag{3.5}$$

How *tight* is the bound of Inequality 3.5? Clearly, it cannot in general be tight for all instances I generated by a recurrent task system τ, since τ may generate different instances that have different loads, while the bound of Inequality 3.5 is unable to distinguish between such different instances. However, we do not in general know

beforehand which specific instance τ may generate during run-time; therefore, the bound of Inequality 3.5 must hold for all instances that τ might legally generate. So a more reasonable formulation of the "tightness" question for Inequality 3.5 would be: Is there *some* instance I that could be generated by τ, for which the load bound of Inequality 3.5 is tight?

The answer to this question depends upon the characteristics of the collection of recurrent tasks comprising τ. Informally, the requirement for Inequality 3.5 to represent a tight bound (in the sense discussed above) is that the different tasks comprising τ be completely independent of one another. This requirement is formalized in the *task independence assumptions* (Baruah et al., 1999). We briefly review these independence assumptions below; a more complete discussion may be found in (Baruah et al., 1999).

There are two requirements that are satisfied by systems adhering to the task independence assumptions.

1. *The run-time behavior of a task does not depend upon the behavior of other tasks in the system.* That is, each task is an independent entity, perhaps driven by separate external events. It is not permissible for one task to generate a job directly in response to another task generating a job. Instances of task systems not satisfying this assumption include systems where, for example, all tasks are required to generate jobs at the same time instant, or where it is guaranteed that certain tasks will generate jobs before certain other tasks. (However, such systems can sometimes nevertheless be represented in such a manner as to satisfy this assumption, by modelling the interacting tasks as a single task which is assumed to generate the jobs actually generated by the interacting tasks.)

2. *The workload constraints can be specified without making any references to "absolute" time.* That is, specifications such as "Task τ_i generates a job at time 3"

are forbidden. There are several scenarios within which this assumption holds. Consider first a distributed system in which each task executes on a separate node (jobs correspond to requests for time on a shared resource) and which begins execution in response to an external event. All temporal specifications are made relative to the time at which the task begins execution, which is not *a priori* known. As another example, consider a distributed system in which each task maintains its own (very accurate) clock, and in which the clocks of different tasks are not synchronized with each other. The accuracy of the clocks permits us to assume that there is no clock drift, and that all tasks use exactly the same units for measuring time. However, the fact that these clocks are not synchronized rules out the use of a concept of an absolute time scale.

These task independence assumptions are extremely general and are satisfied by a wide variety of the kinds of task systems one may encounter in practice. Most common task models, including the LL (Liu and Layland, 1973), multiframe (Mok and Chen, 1996; Mok and Chen, 1997) generalized multiframe (Baruah et al., 1999), and the DAG-based model (Baruah, 2003), satisfy these assumptions.

We now return to the issue of the *tightness* of the bound of Inequality 3.5. Let τ denote a real-time system. If τ satisfies the task independence assumptions, then the bound of Inequality 3.5 is tight in the sense that there is some real-time instance I that could legally be generated by τ, for which

$$\mathsf{load}(I) = \max_{t \geq 0} \left\{ \frac{\sum_{\tau_i \in \tau} \mathrm{DBF}(\tau_i, t)}{t} \right\} .$$

Using this notion of maximum load real-time instance generated by τ, We can in fact define load in terms of the partially specified task system:

Definition 3.5 (load for Task System τ) *The* load *of a recurrent partially-specified task system τ in task model* M *is equal to the maximum load of any real-time instance generated by the task system τ:*

$$\mathsf{load}(\tau) \stackrel{def}{=} \max_{I \in \mathcal{I}^{\mathrm{M}}(\tau)} \{\mathsf{load}(I)\}. \tag{3.6}$$

§ Time Complexity. We now discuss the computational complexity of determining bounds on max-job-density(I) and load(I), when we are given that I is generated by a system τ of recurrent real-time tasks. From the definition of max-job-density (Equation 3.2), it follows that max-job-density(I) is easily bound for any system in which all possible jobs that could be generated by each task can be enumerated; the computational complexity of doing so is directly proportional to the computational complexity of the enumeration[1]. For example, in the sporadic task model, each job J_i generated by sporadic task τ_k has $\frac{E_i}{D_i} = \frac{e_k}{d_k}$; therefore, to determine max-job-density(I), we find the value of $\max_{\tau_k \in \tau}\{e_k/d_k\}$ which clearly has complexity $\mathcal{O}(n)$.

The determination of load(I) is somewhat more complex. From Inequality 3.5, it can be seen that load(I) is defined in terms of the DBF functions of all the tasks comprising τ. It has been shown (Chakraborty et al., 2001; Chakraborty, 2003) that DBFs can be efficiently computed for all the formal models of recurrent tasks (the LL (Liu and Layland, 1973), the sporadic (Mok, 1983), the multiframe (Mok and Chen, 1996; Mok and Chen, 1997), the generalized multiframe (Baruah et al., 1999), and the recurring real-time task model (Baruah, 2003)) discussed in this dissertation. This is achieved by doing a pseudo-polynomial amount of pre-processing per task, after which $\mathrm{DBF}(\tau_i, t)$ for any t can be completed in polynomial time.

[1]When the jobs are characterized by *upper bounds* on their execution requirements, the value of max-job-density(I) so computed also becomes an upper bound.

To implement any of the (sufficient) multiprocessor feasibility test that are based on demand-based load and maximum job density (presented in the future chapters this dissertation) upon a task system τ, we would therefore do the following

1. Perform the DBFpreprocessing on each task in the task system τ.

2. Compute a bound on $\mathsf{max\text{-}job\text{-}density}(I)$.

3. Based upon the computed bound on $\mathsf{max\text{-}job\text{-}density}(I)$ and the available number of processors m, determine a bound B on $\mathsf{load}(I)$ that is implied by a feasibility or schedulability test (for example, Equation 4.1 of Theorem 4.1, or any other test presented in this dissertation).

4. The question now becomes: Is there a $t \geq 0$ such that

$$\sum_{\tau_i \in \tau} \text{DBF}(\tau_i, t) > (B \times t) \ ? \tag{3.7}$$

   If the answer is "no," then τ is guaranteed to be feasible on the m unit-capacity processors. On the other hand, if the answer is "yes" then our test is not able to conclude the feasibility or otherwise of τ on the m processors.

For all the formal models of recurrent tasks considered in this dissertation, it can be shown that if Inequality 3.7 is to be satisfied at all, it will be satisfied for some "reasonably small" value of t — specifically, for some t with value that is no more than pseudo-polynomial in the parameters of the task system (see (Baruah, 2003)). Consequently, we can simply check Inequality 3.7 for all values of t, up to this pseudo-polynomial bound, at which $\text{DBF}(\tau_i, t)$ changes value for some $\tau_i \in \tau$; if Inequality 3.7 is not satisfied for all of these values of t, we can conclude that it will not be satisfied for any value of t, and that τ is consequently feasible.

Recent work (Albers and Slomka, 2004; Baruah and Fisher, 2005b; Fisher and Baruah, 2005a; Fisher and Baruah, 2005b) on *approximation algorithms* for uniprocessor scheduling has introduced many techniques for obtaining polynomial-time feasibility tests for systems of recurrent real-time tasks by "sacrificing" a (quantifiable) fraction of the computing capacity of the available computing capacity. In essence, these techniques approximate the values of $\textsc{dbf}(\tau_i, t)$ beyond a certain (small) value of t. We observe that these techniques all apply to our multiprocessor feasibility test as well; hence, a less accurate variant of our test can be devised, with a run-time complexity that is polynomial in the representation of the task system.

An alternative approach to evaluating Equation 3.7 for determining the feasibility or schedulability of a task system is to pre-compute $\mathsf{load}(\tau)$ for task system τ; then, the validation test is equivalent to checking if $\mathsf{load}(\tau) > B$. In some cases, this may be more computationally expensive, since it may require DBF to be evaluated at a larger number of values than the approach of Equation 3.7. However, for some models such as the sporadic task model, it has been shown that load may be efficiently determined. (Baruah et al., 1990a; Ripoll et al., 1996) present algorithms that have pseudo-polynomial time complexity for task systems that have a system utilization strictly less than m. The next section presents additional algorithms for exactly computing load for sporadic task systems that has pseudo-polynomial time complexity; a PTAS is also presented for approximating load to within an arbitrary additive error term ϵ.

3.4 Efficiently Calculating load for Sporadic Task Systems

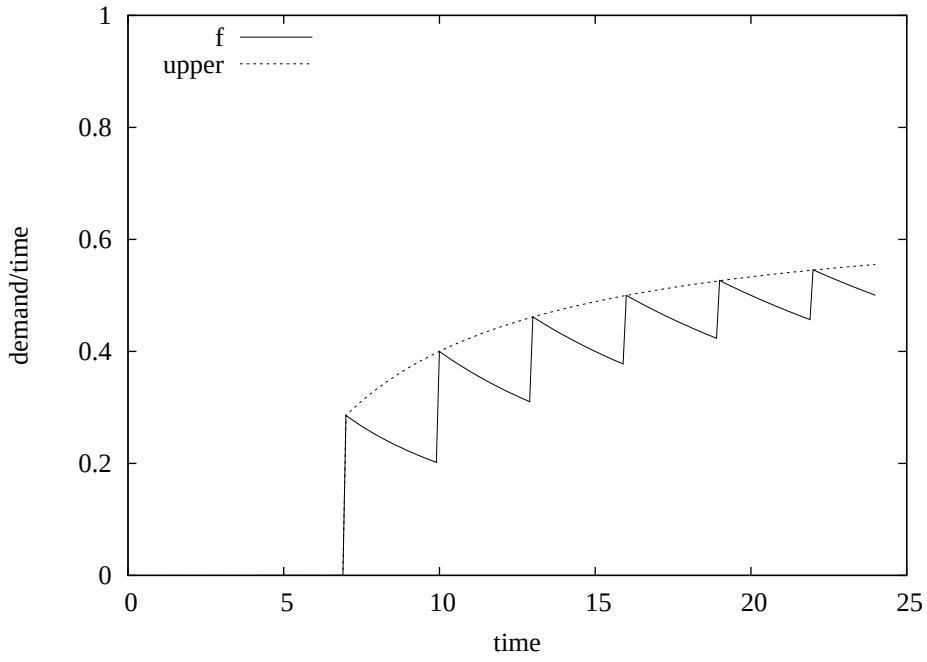
In this section, we address the issue of effectively calculating $\mathsf{load}(\tau)$ if τ is a sporadic task system. We present an exact algorithm for computing $\mathsf{load}(\tau)$ that involves simulating the scheduling of τ to its *hyperperiod* (the least common multiple of the task system's periods, $LCM_{i=1}^{n} p_i$). We also present two other algorithms which approximate $\mathsf{load}(\tau)$ within an arbitrary threshold $\epsilon > 0$ of its exact value. The first approximate algorithm runs in time pseudo-polynomial in the representation of the task system; the second algorithm, in time polynomial in the representation of the task system.

The remainder of this section is organized as follows. We define some additional notation used throughout this section and prove useful properties of $\mathsf{load}(\tau)$ in Section 3.4.1. We then derive a simple method for exactly determining $\mathsf{load}(\tau)$ in Section 3.4.2. We describe the more practical approximations of load in Section 3.4.3.

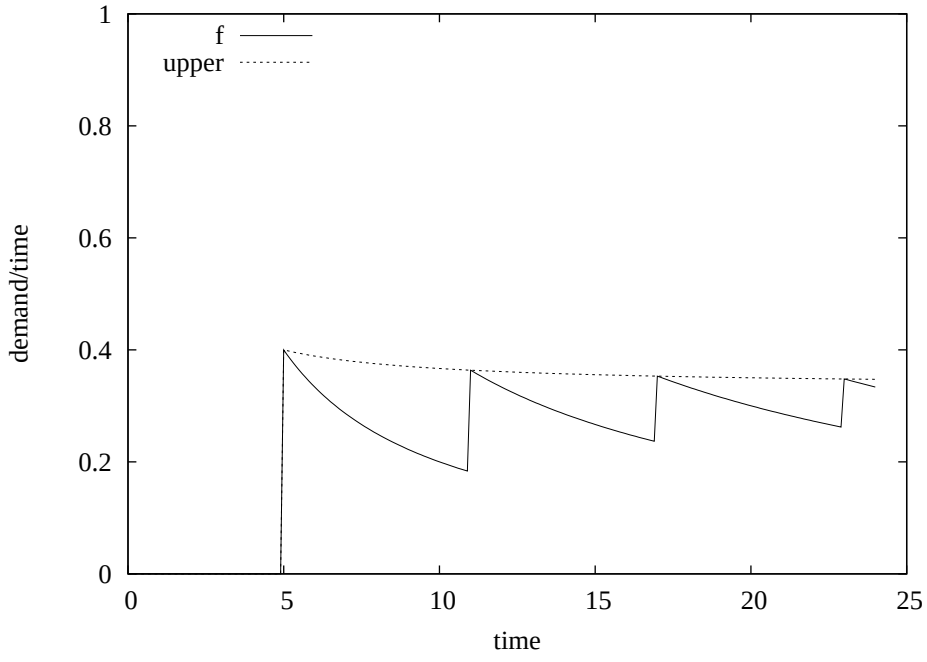
3.4.1 Properties of Demand-Based Load for a Sporadic Task System

Throughout this section, let $f(\tau_i, t)$ be defined to be $\text{DBF}(\tau_i, t)$ normalized by the interval length: $f(\tau_i, t) \stackrel{\text{def}}{=} \dfrac{\text{DBF}(\tau_i, t)}{t}$.

Given a sporadic task system $\tau = \{\tau_1, \ldots, \tau_n\}$, the demand-based load $\mathsf{load}(\tau)$ can be computed by determining $\max_{t>0} f(\tau, t)$ where $f(\tau, t) \stackrel{\text{def}}{=} \sum_{i=1}^{n} f(\tau_i, t)$.
The parameter $\mathsf{load}(\tau)$ may be calculated using the above function because $\text{DBF}(\tau_i, t)$ is a "tight" characterization (per the discussion of the previous subsection and (Baruah et al., 1990a)) for sporadic task systems of the maximum demand of any real-time instance over any interval of length t.



(a)



(b)

Figure 3.1: The zigzag solid lines in both graphs represent values $f(\tau_i, t)$ with respect to t where in (a) $\tau_i \stackrel{\text{def}}{=} (e_i, d_i, p_i) = (2, 7, 3)$, and in (b) $\tau_i \stackrel{\text{def}}{=} (2, 5, 6)$. The dotted lines represent approximations to $f(\tau_i, t)$ with $k_i = 0$. (Approximations are described in Section 3.4.3.2.)

Figure 3.1 illustrates an example of $f(\tau_i, t)$ for two different tasks. The demand-based load, $\mathsf{load}(\tau)$, describes the maximum value of $f(\tau, t)$ over all positive values of t. Since the values of t are real and unbounded, the notation $\max_{t>0}$ here denotes the least upper bound of $f(\tau, t)$.

We will now show that $\mathsf{load}(\tau)$ is potentially superior to $\mathsf{system\text{-}util}(\tau)$ as a lower bound on computational load, as it falls between $\mathsf{system\text{-}util}(\tau)$ and $\mathsf{system\text{-}density}(\tau)$. The next lemma proves that $\mathsf{system\text{-}util}(\tau)$ is a lower bound on the demand-based load.

Lemma 3.2 $\mathsf{load}(\tau) \geq \mathsf{system\text{-}util}(\tau)$

Proof: Observe that for each task $\tau_i \in \tau$, $\lim_{t\to\infty} f(\tau_i, t) = u_i$.

More precisely, for a given i and t, let $0 \leq r < p_i$ be the value such that $\lfloor \frac{t-d_i}{p_i} \rfloor = \frac{t-d_i-r}{p_i}$. It follows that

$$\begin{aligned}\frac{(\lfloor \frac{t-d_i}{p_i} \rfloor + 1)e_i}{t} &= \frac{(\frac{t-d_i-r}{p_i} + 1)e_i}{t} \\ &= \frac{(t + p_i - d_i - r)e_i}{tp_i} \\ &= u_i + u_i \frac{p_i - d_i - r}{t}.\end{aligned}$$

Since $-d_i < p_i - d_i - r < p_i - d_i$, the absolute value of the fraction on the right above is decreasing with respect to t, and so the limit of the entire expression is u_i. It follows that $\lim_{t\to\infty} f(\tau, t) = \mathsf{system\text{-}util}(\tau)$. The lemma immediately follows from this limit. ■

The following lemma shows that $\mathsf{system\text{-}density}(\tau)$ is an upper bound on the demand-based load.

Lemma 3.3 $\mathsf{load}(\tau) \leq \mathsf{system\text{-}density}(\tau)$

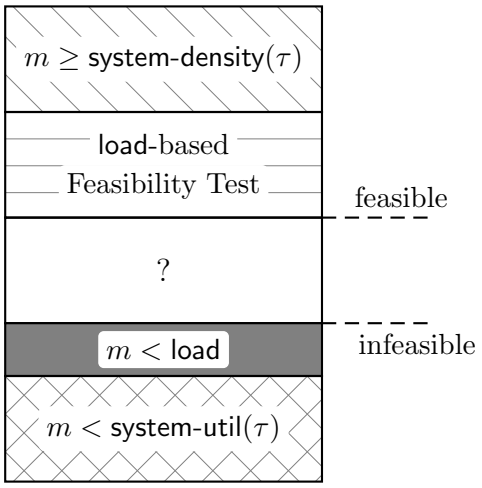


Figure 3.2: load-based tests further reduce the region of uncertainty from Figure 2.2 for the feasibility of sporadic task systems on multiprocessor platforms.

Proof: Note that, by definition of $f(\tau, t)$,

$$f(\tau,t) = \sum_{i:d_i<t} \frac{\lfloor \frac{t+p_i-d_i}{p_i} \rfloor e_i}{t} \leq \sum_{i:d_i<t} u_i \left(1 + \frac{p_i - d_i}{t}\right).$$

If $p_i \geq d_i$, the term $\frac{p_i-d_i}{t}$ is non-increasing with respect to t, and since $d_i < t$ (by the summation terms), $\frac{p_i-d_i}{t} \leq \frac{p_i-d_i}{d_i} = \frac{p_i}{d_i} - 1$.

Otherwise, the term $\frac{p_i-d_i}{t}$ is increasing with respect to t, and in the limit $\frac{p_i-d_i}{t} = 0$.

Therefore,

$$\begin{aligned} f(\tau,t) &\leq \sum_{i:d_i<t} u_i(1 + \max(0, \frac{p_i}{d_i} - 1)) = \sum_{i:d_i<t} \text{task-density}(\tau_i) \\ &\leq \sum_{i=1}^{n} \text{task-density}(\tau_i) = \text{system-density}(\tau). \end{aligned}$$

■

The benefit that load provides for infeasibility and feasibility tests due to Lemmas 3.2 and 3.3 is conceptually represented in Figure 3.2.

3.4.2 An Exact Algorithm for Calculating Load

To calculate $\mathsf{load}(\tau)$, we must limit the number of values of t for which we evaluate $f(\tau, t)$ to a finite number. It may seem that $f(\tau, t)$ needs to be checked at an infinite number of t values. However, the following two observations are useful in showing that only a finite number of values need to be checked.

1. **The maximum value of $f(\tau, t)$ only occurs at "step" points** (Lemma 3.4). Therefore, the set of potential test points is countable.

2. **$f(\tau, t)$ is maximized prior to τ's hyperperiod** (Lemma 3.5). Therefore, the maximum test point has a bounded value.

The following lemma formally restates and proves the first observation.

Lemma 3.4

$$\max_{t>0} f(\tau, t) = \max\{f(\tau, jp_i + d_i) \mid i = 1, \ldots, n;\ j = 0, \ldots\}$$

Proof: Since $f(\tau, t)$ is generally locally decreasing with respect to t, attention can be limited to the values of t for which the derivative is discontinuous, i.e., $t = jp_i + d_i$ for positive integer values j. ■

We may now show that the hyperperiod provides an upper bound on the maximum possible t that we need to evaluate $f(\tau, t)$.

Lemma 3.5 *Let* $L = LCM_{i=1}^{n} p_i$. *If* $\mathsf{load}(\tau) > \mathsf{system\text{-}util}(\tau)$ *then* $\mathsf{load}(\tau) = f(\tau, t)$ *for some* $t \leq L$.

Proof: The proof is by contradiction. Let $cL+x$ be the least value (where $c \geq 1$ and $0 < x < L$) for which $f(\tau, cL+x) = \mathsf{load}(\tau) > \mathsf{system\text{-}util}(\tau)$ and $f(\tau, cL+x) > f(\tau, t)$ for every $t < cL + x$. If the lemma is false, there must be such a value.

By definition of function f,

$$\begin{aligned} f(\tau, cL+x) &= \frac{\sum_{i=1}^{n} \mathrm{DBF}(\tau_i, cL+x)}{cL+x} \\ &= \frac{\sum_{i=1}^{n} \max\left(0, \left(\left\lfloor \frac{cL+x-d_i}{p_i} \right\rfloor + 1\right) \times e_i\right)}{cL+x} \\ &= \frac{\sum_{i=1}^{n} \max\left(0, \left(\frac{cL}{p_i} + \left\lfloor \frac{x-d_i}{p_i} \right\rfloor + 1\right) \times e_i\right)}{cL+x} \\ &\leq \frac{cL \times \mathsf{system\text{-}util}(\tau) + \sum_{i=1}^{n} \mathrm{DBF}(\tau_i, x)}{cL+x}. \end{aligned}$$

Since $f(\tau, cL+x)$ equals $\mathsf{load}(\tau)$, the above equations imply that $cL \times \mathsf{system\text{-}util}(\tau)$ $+ \sum_{i=1}^{n} \mathrm{DBF}(\tau_i, x)$ is at least $(\mathsf{load}(\tau) \times cL) + (\mathsf{load}(\tau) \times x)$. Due to the assumption that $\mathsf{load}(\tau)$ exceeds $\mathsf{system\text{-}util}(\tau)$,

$$\begin{aligned} & (cL \times \mathsf{system\text{-}util}(\tau)) + \textstyle\sum_{i=1}^{n} \mathrm{DBF}(\tau_i, x) && > (cL \times \mathsf{system\text{-}util}(\tau)) + (\mathsf{load}(\tau) \times x) \\ \Rightarrow \quad & \textstyle\sum_{i=1}^{n} \mathrm{DBF}(\tau_i, x) && > (\mathsf{load}(\tau) \times x) \\ \Rightarrow \quad & f(\tau, x) && > \mathsf{load}(\tau). \end{aligned}$$

The last inequality contradicts our supposition that for all $t < cL + x$, $f(\tau, t) < f(\tau, cL + x)$. Thus, the lemma holds. ■

The following corollary to Lemma 3.5 and Lemma 3.2 shows that if $f(\tau, t)$ does not exceed $\mathsf{system\text{-}util}(\tau)$ prior to the hyperperiod of τ, we may infer that $\mathsf{load}(\tau) = \mathsf{system\text{-}util}(\tau)$.

Corollary 3.1 *If* $f(\tau, t) \leq \mathsf{system\text{-}util}(\tau)$ *for all* $t \leq L$, *then* $\mathsf{load}(\tau) = \mathsf{system\text{-}util}(\tau)$.

Lemmas 3.4 and 3.5 and Corollary 3.1 imply that we need to only check $f(\tau, t)$ up to the task system's hyperperiod. We can further limit the number of values t that need to be considered in the exact computation of load if we iteratively make use of the information we have gained by looking at the value of $f(\tau, t)$ up to any given point in the computation. The next lemma formalizes this concept.

Lemma 3.6 *If* $f(\tau, t) \geq \mathsf{system\text{-}util}(\tau) + \gamma$ *for some* $\gamma > 0$ *then* $t \leq \mathsf{system\text{-}util}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\gamma}$.

Proof: Observe that $\max_{\tau_i \in \tau}(p_i - d_i) > 0$; otherwise, for all $\tau_i \in \tau$,

$$\begin{aligned}
\left(\left\lfloor \frac{t - d_i}{p_i} \right\rfloor + 1\right) e_i \quad &\leq \left(\tfrac{t-d_i}{p_i} + 1\right) e_i \\
&\leq u_i t - u_i d_i + u_i p_i \quad \leq u_i t \\
\Rightarrow \qquad & \mathrm{DBF}(\tau_i, t) \qquad \leq u_i t \\
\Rightarrow \qquad & f(\tau, t) \qquad \leq \mathsf{system\text{-}util}(\tau).
\end{aligned}$$

The last statement contradicts the supposition of the lemma. Thus, there must exist a $\tau_i \in \tau$ such that $p_i - d_i > 0$. Therefore,

$$\begin{aligned}
\mathsf{system\text{-}util}(\tau) + \gamma \quad &\leq \quad f(\tau, t) = \frac{\sum_{i=1}^{n} \max(0, (\lfloor \frac{t-d_i}{p_i} \rfloor + 1)e_i)}{t} \\
&\leq \quad \sum_{i:d_i<t} u_i + \sum_{i:d_i<t} u_i \frac{\max_{\tau_j \in \tau}(p_j - d_j)}{t} \\
&\leq \quad \mathsf{system\text{-}util}(\tau)\left(1 + \frac{\max_{\tau_i \in \tau}(p_i - d_i)}{t}\right) \\
\Rightarrow \gamma \quad &\leq \quad \mathsf{system\text{-}util}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{t} \\
\Rightarrow t \quad &\leq \quad \mathsf{system\text{-}util}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\gamma}.
\end{aligned}$$

■

CALCULATE–load(τ, ϵ)

▷ Interpret a divide by zero, as infinity; ϵ can be zero for exact-case.

1 limit $\leftarrow \min\left\{(LCM_{i=1}^{n} p_i), \left(\frac{\sum_{i=1}^{n} e_i}{\epsilon}\right), \left(\text{system-util}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\epsilon}\right)\right\}$

2 fmax $\leftarrow$ system-util(τ);

3 **for** each $t = jp_i + d_i$, in increasing order, **loop**

exit when $t \geq$ limit;

4 **if** $f(\tau, t) >$ fmax

then

5 fmax $\leftarrow f(\tau, t)$;

6 limit $\leftarrow \min(\text{limit}, \text{system-util}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\text{fmax} - \text{system-util}_{(\tau)+\epsilon}})$

7 **if** fmax$>$ system-density$(\tau) - \epsilon$ **then return** system-density(τ);

8 **end if**;

9 **end loop**;

10 **return** fmax;

Figure 3.3: Pseudo-code for determining demand-based load within a value of ϵ. When ϵ equals zero, the algorithm calculates the exact value of the demand-based load; otherwise, it is an approximation.

Our algorithm for calculating load(τ) is represented in Figure 3.3 by CALCULATE–load. The subroutine exactly calculates load(τ) when passed an ϵ parameter equal to zero; note that when $\epsilon = 0$, Line 1 always sets `limit` to the hyperperiod, and Line 6 never updates `limit` (non-zero ϵ values will be discussed in Section 3.4.3.1). Lemmas 3.4, 3.5, and 3.6, and Corollary 3.1 show that CALCULATE–load is correct when $\epsilon = 0$.

§ **Time Complexity.** The "worst-case scenario" with respect to CALCULATE–load$(\tau, 0)$'s execution time occurs when load$(\tau) =$ system-util(τ); in this case, $f(\tau, t) \leq$ system-util(τ) for all $t > 0$. Therefore, `fmax` $\leq$ system-util(τ) for all iterations of CALCULATE–load$(\tau, 0)$. Observe that Line 6 of CALCULATE–load updates `limit` only if `fmax` $-$ system-util$(\tau) > 0$ (assuming $\epsilon = 0$). Consequentially, `limit` is never updated after Line 1 sets it to τ's hyperperiod, and the algorithm calculates $f(\tau, t)$ for all integer values in the task system's hyperperiod. If $p_1, p_2, \ldots, p_n$ are

relatively prime and p_i are integers greater than 1, then $LCM_{i=1}^{n} p_i = \prod_{i=1}^{n} p_i \geq 2^n$. Therefore, the number of time $f(\tau, t)$ is evaluated in CALCULATE–$\mathsf{load}(\tau, 0)$ is potentially exponential in the number of tasks in the task system.

3.4.3 Approximation Algorithms

We can accelerate the convergence of our demand-based load calculation if we permit a bounded level of inaccuracy in our calculation. That is, we can significantly reduce the number of values of t we must consider if we allow our calculated value of $\mathsf{load}(\tau)$ to lie within a specified range (or "tolerance") of the actual value.

Let ϵ denote a tolerance within which $\mathsf{load}(\tau)$ is to be approximated, for arbitrary $\epsilon > 0$. In this section, we propose two algorithms that calculate $\mathsf{load}(\tau)$ to within an additive error ϵ of its actual value. The first algorithm, discussed in Section 3.4.3.1, is a pseudo-polynomial-time algorithm based on the idea of iterative convergence. The second algorithm, presented in Section 3.4.3.2, is a polynomial-time approximation scheme for determining load.

3.4.3.1 Pseudo-polynomial-time Approximation Scheme

The effect of introducing a bounded amount of inaccuracy allows us to limit the number of values of t at which we evaluate $f(\tau, t)$. A useful observation is that after a sufficiently large value of t, $f(\tau, t)$ does not exceed $\mathsf{system\text{-}util}(\tau)$ by more than ϵ. The next lemma quantifies the value of t for which $f(\tau, t)$ is within ϵ of $\mathsf{system\text{-}util}(\tau)$.

Lemma 3.7 *If* $t \geq \frac{\sum_{i=1}^{n} e_i}{\epsilon}$, *then* $f(\tau, t) \leq \mathsf{system\text{-}util}(\tau) + \epsilon$.

Proof: For any $\tau_i \in \tau$ with $t \geq d_i$, $\text{DBF}(\tau_i, t) = (\lfloor \frac{t-d_i}{p_i} \rfloor + 1)e_i$; otherwise, if $t < d_i$, $\text{DBF}(\tau_i, t) = 0$. Since $\lfloor x \rfloor \leq x$, the function $f(\tau, t)$ can be bounded above as follows:

$$f(\tau, t) \leq \sum_{i:d_i<t} \frac{\frac{t-d_i}{p_i}e_i + e_i}{t} \leq \sum_{i:d_i<t} u_i(1 - \frac{d_i}{t}) + \frac{\sum_{i=1}^{n} e_i}{t}$$

$$\Rightarrow f(\tau, t) \leq \mathsf{system\text{-}util}(\tau) - \frac{\sum_{i:d_i<t} u_i d_i}{t} + \frac{\sum_{i=1}^{n} e_i}{t}.$$

It follows that for $t \geq \frac{\sum_{i=1}^{n} e_i}{\epsilon}$,

$$f(\tau, t) \leq \mathsf{system\text{-}util}(\tau) - \frac{\sum_{i:d_i<t} u_i d_i}{t} + \epsilon.$$

Since $-\frac{\sum_{i:d_i<t} u_i d_i}{t} < 0$, for all $t > \frac{\sum_{i=1}^{n} e_i}{\epsilon}$ the following condition is true: $f(\tau, t) \leq \mathsf{system\text{-}util}(\tau) + \epsilon$. ■

Since `fmax` never decreases and is initially $\mathsf{system\text{-}util}(\tau)$ in CALCULATE–$\mathsf{load}(\tau, \epsilon)$, Lemma 3.7 implies that we need only evaluate values of $t \leq \frac{\sum_{i=1}^{n} e_i}{\epsilon}$. This optimization is reflected in Line 1 of CALCULATE–$\mathsf{load}(\tau, \epsilon)$.

In addition, we can use Lemma 3.6 to further reduce the number of steps taken by CALCULATE–$\mathsf{load}(\tau, \epsilon)$. Suppose at step i in an iterative approximate computation of $\mathsf{load}(\tau)$ the maximum value of $f(\tau, t)$ has been computed over all values $t \leq t_i$, and that value is `fmax`. The computation can terminate unless there is a value $t > t_0$ such that $f(\tau, t) > f(\tau, t_0) + \epsilon$. Letting $\gamma = \texttt{fmax} - \mathsf{system\text{-}util}(\tau) + \epsilon$ in Lemma 3.6 above we have

$$t \leq \mathsf{system\text{-}util}(\tau) \frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\text{fmax} - \mathsf{system\text{-}util}(\tau) + \epsilon}.$$

The above observations are applied in Lines 1 and 6 of the algorithm CALCULATE–$\mathsf{load}(\tau, \epsilon)$ for approximate computation of $\mathsf{load}(\tau)$ (shown in Figure 3.3). Line 7 allows the algorithm to terminate early if `fmax` is within ϵ of our upper bound of $\mathsf{system\text{-}density}(\tau)$.

§ **Time Complexity.** The values of t for which we must evaluate $f(\tau, t)$ in CALCULATE–$\mathsf{load}(\tau, \epsilon)$ is at most $\min\left\{\left(\frac{\sum_{i=1}^{n} e_i}{\epsilon}\right), \left(\mathsf{system\text{-}util}(\tau)\frac{\max_{\tau_i \in \tau}(p_i - d_i)}{\epsilon}\right)\right\}$. Both of these terms are polynomial in the task system's parameters and $1/\epsilon$, and each evaluation of $f(\tau, t)$ requires $\mathcal{O}(n)$ time. Therefore, CALCULATE–$\mathsf{load}(\tau, \epsilon)$ represents a *pseudo-polynomial approximation scheme* (PPTAS).

3.4.3.2 Polynomial-time Approximation Scheme

Further theoretical reduction in the number of potential values for which to evaluate $f(\tau, t)$ can be achieved by an approximation to $\text{DBF}(\tau_i, t)$. Our approximation will allow us to "skip" intermediate test points in the calculation of $\mathsf{load}(\tau)$. We may approximate $\text{DBF}(\tau_i, t)$ for each task τ_i by "tracking" it exactly for $k_i + 1$ steps (how to pick k_i will be discussed shortly), and then using the tightest linear upper bound of $\text{DBF}(\tau_i, t)$ with slope u_i after $k_i + 1$ steps. A similar approximation was defined by Albers and Slomka (Albers and Slomka, 2004) (based on an approximation introduced by (Devi, 2003)) for uniprocessor feasibility analysis. Formally, the approximation can be expressed by:

$$\text{DBF}^*(\tau_i, t, k_i) \stackrel{\text{def}}{=} \begin{cases} \text{DBF}(\tau_i, t), & \text{if } t < k_i p_i + d_i, \\ e_i + (t - d_i)u_i, & \text{otherwise.} \end{cases} \tag{3.8}$$

We can now describe approximations to $\mathsf{load}(\tau)$ using $\text{DBF}^*(\tau_i, t, k_i)$:

$$\begin{aligned} f^*(\tau_i, t) &\stackrel{\text{def}}{=} \frac{\text{DBF}^*(\tau_i, t, k_i)}{t}, \\ f^*(\tau, t) &\stackrel{\text{def}}{=} \sum_{i=1}^{n} f^*(\tau_i, t), \end{aligned}$$

and

$$\mathsf{load}^*(\tau) \stackrel{\text{def}}{=} \max_{t>0} f^*(\tau, t).$$

Visual examples of $f^*(\tau_i, t)$ are shown in Figure 3.1 with $k_i = 0$.

It can be shown that if we pick $k_i \stackrel{\text{def}}{=} \max\left(\lceil \frac{nu_i}{\epsilon} - \frac{d_i}{p_i} \rceil, 0\right)$, then $\mathsf{load}^*(\tau)$ is within ϵ of $\mathsf{load}(\tau)$. The following lemma proves this assertion.

Lemma 3.8 *For sporadic task system τ, if for all $\tau_i \in \tau$, $k_i = \max\left(\lceil \frac{nu_i}{\epsilon} - \frac{d_i}{p_i} \rceil, 0\right)$, then $\mathsf{load}(\tau) \leq \mathsf{load}^*(\tau) \leq \mathsf{load}(\tau) + \epsilon$.*

Proof: To prove the lemma it suffices to show for all $t > 0$ that $f(\tau, t) \leq f^*(\tau, t) \leq f(\tau, t) + \epsilon$. Obviously, $f(\tau, t) \leq f^*(\tau, t)$; so, we will focus on showing that $f^*(\tau, t) \leq f(\tau, t) + \epsilon$ for the k_i specified in the lemma.

Consider the following partition of $\tau = \tau_{\text{dbf}-\text{exact}}(t) \cup \tau_{\text{dbf}-\text{approx}}(t)$ where $\tau_{\text{dbf}-\text{approx}}(t) \stackrel{\text{def}}{=} \{\tau_i | k_i p_i + d_i \leq t\}$ and $\tau_{\text{dbf}-\text{exact}}(t) \stackrel{\text{def}}{=} \tau - \tau_{\text{dbf}-\text{approx}}(t)$. Informally, $\tau_{\text{dbf}-\text{exact}}(t)$ is the set of tasks that have not taken $k_i + 1$ exact steps of DBF^* at time t; and, $\tau_{\text{dbf}-\text{approx}}(t)$ is the set of tasks where $\text{DBF}^*(\tau_i, t, k_i)$ uses the linear approximation to $\text{DBF}(\tau_i, t)$ at time t.

Observe that $\text{DBF}^*(\tau_i, t, k_i) \leq \text{DBF}(\tau_i, t) + e_i$ for all $t > 0$ and $\tau_i \in \tau$. Therefore,

$$\begin{aligned} f^*(\tau, t) &= \sum_{\tau_i \in \tau_{\text{dbf}-\text{exact}}} \frac{\text{DBF}(\tau_i, t)}{t} \\ &\quad + \sum_{\tau_i \in \tau_{\text{dbf}-\text{approx}}} \frac{\text{DBF}^*(\tau_i, t, k_i)}{t} \\ &\leq \sum_{i=1}^{n} \frac{\text{DBF}(\tau_i, t)}{t} + \sum_{\tau_i \in \tau_{\text{dbf}-\text{approx}}} \frac{e_i}{t} \\ &= f(\tau, t) + \sum_{\tau_i \in \tau_{\text{dbf}-\text{approx}}} \frac{e_i}{t}. \end{aligned}$$

Note that for all $\tau_i \in \tau_{\text{dbf}-\text{approx}}$, the value of t is lower bounded by $k_i p_i + d_i$. This implies $t \geq \left(\frac{nu_i}{\epsilon} - \frac{d_i}{p_i}\right) p_i + d_i = \frac{ne_i}{\epsilon}$. Thus,

$$\begin{aligned} f^*(\tau, t) &\leq f(\tau, t) + \sum_{\tau_i \in \tau_{\text{dbf}-\text{approx}}} \frac{\epsilon}{n} \\ &\leq f(\tau, t) + \epsilon. \end{aligned}$$

■

The immediate implication of the previous lemma is that we may approximate $\mathsf{load}(\tau)$ by effectively computing $\mathsf{load}^*(\tau)$.

Informally, to determine $\mathsf{load}^*(\tau)$ we need to only evaluate $f^*(\tau, t)$ at times t where the derivative of f^* is discontinuous. The points at which such discontinuities occur for a given sporadic task system τ are:

$$\mathcal{S}(\tau, \epsilon) \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} \{jp_i + d_i | i = 1, \ldots, n; j = 0, \ldots, k_i\}.$$

The next lemma formalizes the assertion that to correctly calculate $\mathsf{load}^*(\tau)$ it is sufficient to only evaluate $f^*(\tau, t)$ for values of $t \in \mathcal{S}(\tau, \epsilon)$. Let $t_1, t_2 \in \mathcal{S}(\tau, \epsilon)$ $(t_1 < t_2)$ be *adjacent* if there does not exist a $t' \in \mathcal{S}(\tau, \epsilon)$ such that $t_1 < t' < t_2$.

Lemma 3.9 *Consider any two adjacent elements of $t_1, t_2 \in \mathcal{S}(\tau, \epsilon) \cup \{0\}$ where $t_1 < t_2$; for all t such that $t_1 < t < t_2$, the following condition holds,*

$$f^*(\tau, t) \leq \max(f^*(\tau, t_1), f^*(\tau, t_2), \mathsf{system\text{-}util}(\tau)).$$

Proof: Let $\tau_{\text{dbf}-\text{approx}}(t)$ and $\tau_{\text{dbf}-\text{exact}}(t)$ be the partition of τ defined in Lemma 3.8. Consider any t in the interval (t_1, t_2). Clearly, $\tau_{\text{dbf}-\text{exact}}(t_1) = \tau_{\text{dbf}-\text{exact}}(t)$. Moreover, there does not exist t' in (t_1, t_2), $\tau_i \in \tau_{\text{dbf}-\text{exact}}(t_1)$, and $\ell \in \mathbb{N}^+$ such that $t' = \ell p_i + d_i$. This implies for all $\tau_i \in \tau_{\text{dbf}-\text{exact}}(t)$,

$$\begin{aligned} f^*(\tau_i, t) &= \frac{\text{DBF}(\tau, t_1)}{t} = \frac{\text{DBF}(\tau, t_1)/t_1}{t/t_1} \\ &= \frac{t_1 f^*(\tau_i, t_1)}{t}. \end{aligned}$$

Also, $\tau_{\text{dbf}-\text{approx}}(t_1) = \tau_{\text{dbf}-\text{approx}}(t)$ which implies for all $\tau_i \in \tau_{\text{dbf}-\text{approx}}(t)$ that $f^*(\tau_i, t) = \frac{t_1 f^*(\tau_i, t_1)}{t} + \frac{u_i(t - t_1)}{t}$. So, for t in the interval (t_1, t_2), we may express $f^*(\tau, t)$

in terms of this partition:

$$\begin{aligned} f^*(\tau,t) &= \sum_{\tau_i \in \tau_{\mathrm{dbf-exact}}} f^*(\tau_i,t) \\ &\quad + \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} f^*(\tau_i,t) \\ &= \sum_{\tau_i \in \tau_{\mathrm{dbf-exact}}} \frac{t_1 f^*(\tau_i,t_1)}{t} \\ &\quad + \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} \left[\frac{t_1 f^*(\tau_i,t_1)}{t} + \frac{u_i(t-t_1)}{t} \right] \\ &= \frac{t_1 f^*(\tau,t_1) + (t-t_1) \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} u_i}{t}. \end{aligned}$$

Let us now look at the partial derivative of $f^*(\tau,t)$ with respect to t:

$$\begin{aligned} \frac{\partial f^*(\tau,t)}{\partial t} &= \frac{t \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} u_i}{t^2} \\ &\quad - \frac{\left(t_1 f^*(\tau,t_1) + (t-t_1) \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} u_i \right)}{t^2} \\ &= \frac{t_1 \left(\sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} u_i - f^*(\tau,t_1) \right)}{t^2}. \end{aligned}$$

Therefore, if $f^*(\tau,t_1) \geq \sum_{\tau_i \in \tau_{\mathrm{dbf-approx}}} u_i$, then $f^*(\tau,t)$ is *non-increasing* and $f^*(\tau,t) \leq f^*(\tau,t_1)$. Otherwise, $f^*(\tau,t)$ is bounded from above by $\mathsf{system\text{-}util}(\tau)$. To complete the lemma, consider t in the interval $(0, \min\{\mathcal{S}\})$ (i.e., 0 and $\min\{\mathcal{S}\}$ are adjacent). In this case, $f^*(\tau,t) = 0 \leq f^*(\tau, \min\{\mathcal{S}\})$. ■

Let $t_{max} \stackrel{\text{def}}{=} \max\{t \in \mathcal{S}(\tau,\epsilon)\}$. By the previous lemma, values of t in the interval $(0,t_{max})$ that are not in $\mathcal{S}(\tau,\epsilon)$ do not contribute to the calculation of $\mathsf{load}^*(\tau)$ (i.e., $f^*(\tau,t) \neq \mathsf{load}^*(\tau)$). The next lemma shows that values of t in the interval (t_{max},∞) also do not contribute to $\mathsf{load}^*(\tau)$:

Lemma 3.10 *For all $t > t_{max}$, the following inequality holds* $\max\left(f^*(\tau,t_{max}), \mathsf{system\text{-}util}(\tau)\right) \geq f^*(\tau,t)$.

Proof: Define $\tau_{\mathrm{dbf-approx}}(t_{max})$ as in Lemma 3.8. For all $t > t_{max}$ and $\tau_i \in \tau$,

PTAS–$\mathsf{load}(\tau, \epsilon)$

```
1  fmax ← system-util(τ) + ε;
   ▷ k_i ≝ max(⌈nu_i/ε − d_i/p_i⌉, 0) for all τ_i ∈ τ.
2  for each t ∈ S(τ, ε), in increasing order loop
3          if f*(τ, t) > fmax
             then
4                  fmax ← f*(τ, t);
5                  if fmax≥ system-density(τ) then return system-density(τ);
6          end if;
7  end loop;
8  return fmax;
```

Figure 3.4: Pseudo-code for determining demand-based load within a value of ϵ in polynomial-time.

$t > k_i p_i + d_i$. This implies that $\tau_{\mathrm{dbf-approx}}(t_{max})$ equals τ. Therefore,

$$f^*(\tau, t) = \frac{t_{max} f^*(\tau, t_{max}) + (t - t_{max})\mathsf{system\text{-}util}(\tau)}{t}.$$

If $f^*(\tau, t_{max}) > \mathsf{system\text{-}util}(\tau)$, then $f^*(\tau, t)$ is decreasing in the interval of (t_{max}, ∞) implying $f^*(\tau, t_{max}) \geq f^*(\tau, t)$; otherwise, $f^*(\tau, t) \leq \mathsf{system\text{-}util}(\tau)$. ■

The algorithm PTAS–$\mathsf{load}(\tau, \epsilon)$ is presented in Figure 3.4. Lemma 3.8 showed that to approximate $\mathsf{load}(\tau)$ we could calculate $\mathsf{load}^*(\tau)$ instead (via evaluating $f^*(\tau, t)$). Lemmas 3.9 and 3.10 showed that we only need to consider the values of t in the set $\mathcal{S}(\tau, \epsilon)$. By these lemmas, PTAS–$\mathsf{load}(\tau, \epsilon)$ correctly approximates $\mathsf{load}(\tau)$ to within an additive value of ϵ. It should be noted that the heuristics of Lemmas 3.5, 3.6, and 3.7 can also be applied to PTAS–$\mathsf{load}(\tau, \epsilon)$ to further speed-up computation.

The number of iterations of PTAS–$\mathsf{load}(\tau, \epsilon)$ is entirely based on the size of the set

$\mathcal{S}(\tau, \epsilon)$. The size of the set is:

$$\sum_{i=1}^{n}(k_i + 1) \in \mathcal{O}\left(\frac{n^2}{\epsilon}\right).$$

because $k_i \leq n/\epsilon$. A straightforward implementation of $f^*(\tau, t)$ has $\mathcal{O}(n)$ complexity. Therefore, the total time complexity of PTAS–$\mathsf{load}(\tau, \epsilon)$ is $\mathcal{O}(n^3/\epsilon)$. The runtime of the algorithm is polynomial in the number of tasks and ϵ, and independent of the task parameters. Therefore, PTAS–$\mathsf{load}(\tau, \epsilon)$ represents a polynomial-time approximation scheme.

3.5 Summary

As we observed in Chapter 2, the system utilization and system density parameters of a task system do not effectively characterize the computational demand of a sporadic task system and more general task systems. For the purposes of developing a better characterization, we propose in this chapter using the well-known parameter of load and max-job-density. In this chapter, we showed that these two metrics are closely related to the feasibility of a recurrent task system on a multiprocessor platform — later chapters will provide further support for this claim. Furthermore, we described how both load and max-job-density may be efficient computed for recurrent task systems in general. We described, in the last section of this chapter, how to compute $\mathsf{load}(\tau)$ of a sporadic task system by providing an exact and approximate algorithms. We demonstrated that it is possible to approximate $\mathsf{load}(\tau)$ to within an additive constant $\epsilon > 0$ in polynomial-time.

Chapter 4

The Restricted- and Full-Migration Feasibility Analysis of General Task Systems

Given a real-time instance I, determining whether I is restricted-migration feasible upon a given number of processors is at least as hard as bin-packing (Johnson, 1973), and so is NP-hard in the strong sense. Exponential-time algorithms are known for solving this problem. By applying network flow techniques, migratory feasibility analysis can be performed in time polynomial in the number of jobs in instance I. A migratory, static schedule for instance I may also be obtained from these techniques. However, both the exponential-time restricted-migration feasibility analysis and the polynomial-time full-migration feasibility analysis algorithms require that all parameters of all the jobs in instance I be known beforehand. As discussed in the introduction, this may not always be possible since many real-time applications are comprised of partially-specified recurrent real-time tasks.

In the last chapter, we observed (Lemma 3.2) that load and max-job-density may be used as necessary conditions for the feasibility of real-time instances on multiproces-

sor platforms. In this chapter, we derive conditions based on load and max-job-density that are sufficient for a real-time instance to be feasible upon an m-processor platform. Both restricted-migration and full-migration systems are considered. Furthermore, the feasibility results contained in this chapter have a constant factor resource-augmentation approximation ratio (discussed in the introduction of this dissertation). The feasibility results of this chapter are obtained for real-time instances; by way of Sections 3.3 and 3.4 of the previous chapter, these results are also directly relevant to determining the feasibility of partially-specified recurrent task systems because load and max-job-density may be obtained for these systems. Section 4.1 will derive sufficient feasibility conditions for restricted-migration systems. Section 4.2 will obtain conditions for full-migration systems.

4.1 Restricted-Migration Feasibility

We now derive sufficient conditions for determining whether a given real-time instance I is restricted-migration feasible upon a specified number of processors based on the parameters max-job-density(I) and load(I). Since migratory scheduling is more general than restricted-migration scheduling (in the sense that every restricted-migration schedule is also a migratory schedule in which no migrations happened to occur), these sufficient conditions are sufficient conditions for migratory feasibility analysis, too.

Suppose that a given real-time instance $I = J_1, J_2, \ldots$ is *in*feasible upon m unit-capacity processors. Without loss of generality, let us assume that the jobs are *indexed by non-decreasing order of relative deadline* — i.e., $D_i \leq D_{i+1}$ for all $i \geq 1$ (note, that this is a different order than assumed in the introduction). We now describe an (off-line) multiprocessor algorithm for scheduling this collection of jobs; since the collection of jobs has no schedule (by virtue of being infeasible), it is necessary that at some point during its execution this algorithm will report failure.

Our algorithm considers jobs in the order $J_1, J_2, J_3, \ldots$, i.e., in non-decreasing order of relative deadline. In considering a job J_i, the goal is to assign it to a processor in such a manner that all the jobs assigned to each processor are preemptive uniprocessor feasible.

We now describe in detail the algorithm for assigning job J_i:

- For each processor $\pi_k, 1 \leq k \leq m$, let $I(\pi_k)$ denote the jobs that have already been assigned to this processor; our assignment algorithm ensures that $I(\pi_k)$ is preemptive uniprocessor feasible.

- Assign J_i to any processor π_k such that doing so retains feasibility on that processor; if no such π_k exists, declare failure and exit.

A pseudo-code representation of this job-assignment algorithm is presented in Figure 4.1.

```
JOBASSIGN
    ▷ There are m unit-capacity processors, denoted π_1, π_2, ..., π_m
    ▷ I(π_k) denotes the jobs already assigned to processor π_k
1   for i ← 1, 2, ...
2       if there is a processor π_k such that (I(π_k) ⋃ {J_i}) is preemptive uniprocessor feasible
3       then
                ▷ assign J_i to π_k
4               I(π_k) ← I(π_k) ⋃ {J_i}
5       else return ASSIGNMENT FAILED
```

Figure 4.1: Pseudo-code for job-assignment algorithm.

4.1.1 Algorithm Analysis

Now, we will examine the conditions under which Algorithm JOBASSIGN may fail. By ensuring that these conditions are never satisfied, we can then obtain a sufficient

schedulability test for feasibility analysis of real-time instances.

Theorem 4.1 *Let I denote a real-time instance satisfying*

$$\mathsf{load}(I) \leq \frac{1}{3}\left(m - (m-1)\mathsf{max\text{-}job\text{-}density}(I)\right). \tag{4.1}$$

Algorithm JobAssign *successfully assigns every job to some processor, on a platform comprised of m unit-capacity processors.*

Proof:

Let us assume that Algorithm JobAssign does not successfully assign all jobs in I, and let J_i denote the first job for which Algorithm JobAssign returns ASSIGNMENT FAILED.

For each processor π_k, $I(\pi_k)$ is feasible by construction. Let W_k denote the minimum amount of execution that is performed over the interval $[A_i, A_i + D_i)$ in any preemptive uniprocessor schedule for $I(\pi_k)$ that meets all deadlines. Since J_i cannot be accommodated on any processor (i.e., $I(\pi_k) \cup \{J_i\}$ is infeasible for all processors), it must be the case that $W_k > (D_i - E_i)$ on each processor (otherwise, there would be enough "idle time" with respect to $I(\pi_k)$ to complete J_i's execution by its deadline, and $I(\pi_k) \cup \{J_i\}$ would be feasible on π_k). Summing over all m processors, we conclude that

$$\sum_{k=1}^{m} W_k > m \cdot (D_i - E_i) \tag{4.2}$$

Up until Algorithm JobAssign fails to assign job J_i to a processor, only jobs with relative deadline at most D_i have been assigned to processors. Therefore, no job in $I(\pi_k)$ for any processor π_k can have an arrival time of at most A_i and absolute deadline exceeding $A_i + D_i$; otherwise, such a job would have relative deadline greater than D_i and would contradict the assignment ordering of Algorithm JobAssign. Thus, all

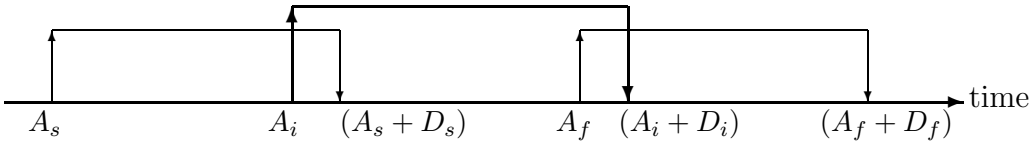


Figure 4.2: (Proof of Theorem 4.1.) Job J_i is being considered. Jobs J_s and J_f are the previously-assigned jobs with earliest arrival time and latest absolute deadline respectively, whose "intervals" overlap with that of J_i.

of the execution of jobs of $I(\pi_k)$ (for any processor π_k) in the interval $[A_i, A_i + D_i)$ belongs to jobs that arrive or have a deadline in the interval $[A_i, A_i + D_i)$.

Let J_s denote the job with earliest arrival time that has already been assigned to some processor, such that $A_s + D_s > A_i$; and let J_f denote the job with latest (absolute) deadline that has already been assigned to some processor, such that $A_f < A_i + D_i$ (see Figure 4.2).

Since jobs are considered in order of their relative deadline and J_s and J_f were both assigned prior to job J_i being considered, it must be the case that D_s and D_f are both no larger than D_i:

$$D_s \leq D_i \text{ and } D_f \leq D_i \ . \tag{4.3}$$

From Figure 4.2, it is immediately evident that the interval $[A_s, A_f + D_f)$ is of size no more than $3 \times D_i$:

$$(A_f + D_f) - A_s \leq 3 \cdot D_i \ . \tag{4.4}$$

Hence, all the work contributing to the expression on the right-hand side of Inequality 4.2 was generated by jobs that both arrive in, and have their deadline within, the interval $[A_s, A_f + D_f)$, which is of length at most $\leq 3 \times D_i$. By the definition of $\mathsf{load}(I)$, the maximum amount of work arriving in, and having deadlines within, an interval of this length is at most $(A_f + D_f - A_s) \times \mathsf{load}(I)$, of which an amount E_i (cor-

responding to the execution requirement of J_i) does not contribute to the right-hand side of Inequality 4.2. Hence,

$$\begin{aligned}
&(A_f + D_f - A_s)\mathsf{load}(I) - E_i > m(D_i - E_i) \\
\Rightarrow \quad &\text{(By Inequality 4.4 above)} \qquad (4.5) \\
&3 \cdot D_i \cdot \mathsf{load}(I) - E_i > m(D_i - E_i) \\
\equiv \quad &\mathsf{load}(I) > \frac{m - (m-1)\ \frac{E_i}{D_i}}{3}. \qquad (4.6)
\end{aligned}$$

Note that the right-hand side decreases as $\frac{E_i}{D_i}$ increases; i.e., this condition is more likely to be satisfied for larger values of $\frac{E_i}{D_i}$. Since a *sufficient* condition for feasibility is that the negation of Condition 4.6 always hold, this is ensured by requiring that the negation of Condition 4.6 hold for the largest possible value of $\frac{E_i}{D_i}$, i.e., for $\frac{E_i}{D_i} = \mathsf{max\text{-}job\text{-}density}(I)$:

$$\mathsf{load}(I) \leq \frac{1}{3}\left(m - (m-1)\ \mathsf{max\text{-}job\text{-}density}(I)\right),$$

which is exactly Inequality 4.1 of the statement of the theorem. ■

Recall that our goal has been to obtain sufficient conditions for preemptive multiprocessor feasibility. Specifically, we had set out to obtain a *feasibility region* in the two-dimensional space $[0, 1] \times [0, m]$, such that any instance I with $\mathsf{max\text{-}job\text{-}density}(I)$ and $\mathsf{load}(I)$ lying in this region is guaranteed to be feasible. Theorem 4.1 yields such a feasibility region: for given m, any instance I satisfying Equation 4.1 is guaranteed to be feasible upon m unit-capacity processors. In fact, it is also known from uniprocessor scheduling theory that any instance I satisfying ($\mathsf{load}(I) \leq 1$ and $\mathsf{max\text{-}job\text{-}density}(I) \leq 1$) is feasible upon a single unit-capacity processor; hence, such an instance I is also feasible upon m unit-capacity processors for all $m > 1$. Thus, we can modify Inequality 4.1 to come up with the following sufficient condition for

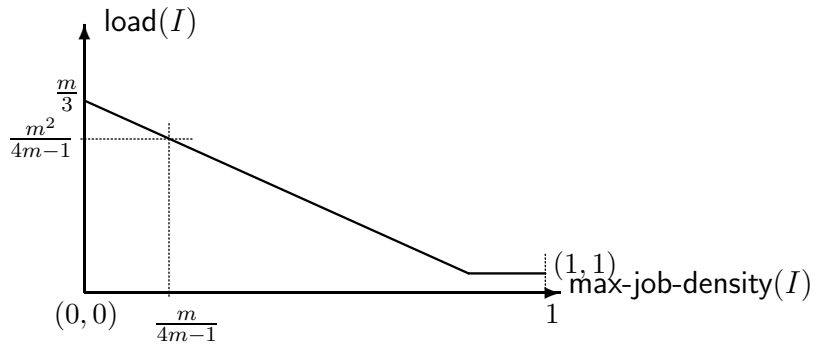


Figure 4.3: The feasibility region, as defined by Equation 4.7. All real-time instances I for which $(\mathsf{max\text{-}job\text{-}density}(I), \mathsf{load}(I))$ lies beneath the solid line are guaranteed feasible on m unit-capacity processors.

feasibility:

$$\mathsf{load}(I) \leq \max\left(1, \frac{1}{3}\left(m - (m-1)\mathsf{max\text{-}job\text{-}density}(I)\right)\right). \tag{4.7}$$

This feasibility region is depicted visually in Figure 4.3.

By Lemma 3.1, recall that a *necessary* condition for instance I to be feasible on m unit-capacity processors is that $\mathsf{load}(I) \leq m$ and $\mathsf{max\text{-}job\text{-}density}(I) \leq 1$; Inequality 4.7 provides sufficient conditions. The following corollary to Theorem 4.1 formalizes this fact:

Corollary 4.1 *Any real-time instance I satisfying the following two conditions*

$$\mathsf{load}(I) \leq \frac{m^2}{4m-1} \tag{4.8}$$

$$\mathsf{max\text{-}job\text{-}density}(I) \leq \frac{m}{4m-1} \tag{4.9}$$

is feasible upon an m-processor unit-capacity multiprocessor platform under restricted-migration multiprocessor scheduling[1]*.*

[1] An alternative statement of this corollary could be: *the point* $(\frac{m}{4m-1}, \frac{m^2}{4m-1})$ *lies in the feasibility region of Figure 4.3* — this point is depicted by the dotted lines in Figure 4.3.

Proof: In order that instance I be feasible on m unit-capacity processors it is, by Equation 4.1, sufficient that

$$\begin{aligned} & \mathsf{load}(I) \leq \frac{1}{3} \times (m - (m-1)\mathsf{max\text{-}job\text{-}density}(I)) \\ \Leftarrow \quad & \text{(From Condition 4.9)} \\ & \mathsf{load}(I) \leq \frac{1}{3} \times \left(m - (m-1)\frac{m}{4m-1} \right) \\ \equiv \quad & \mathsf{load}(I) \leq \frac{m^2}{4m-1}, \end{aligned}$$

which is true (by Condition 4.8 above). ■

The result in Corollary 4.1 above can be considered to be an analog of a "utilization" test (Liu and Layland, 1973), or a "processor demand criterion" test (Baruah et al., 1990b), for uniprocessor systems. According to Lemma 3.1, a necessary condition for real-time instance to be feasible on m unit-capacity processors is that $\mathsf{load}(I) \leq m$ and $\mathsf{max\text{-}job\text{-}density}(I) \leq 1$; by Corollary 4.1 above, it is *sufficient* that $\mathsf{load}(I) \leq m^2/(4m-1)$ and $\mathsf{max\text{-}job\text{-}density}(I) \leq m/(4m-1)$. Hence to within a constant factor of less than four, we have obtained bounds on the values of $\mathsf{load}(I)$ and $\mathsf{max\text{-}job\text{-}density}(I)$ that are needed (and suffice) for feasibility. The resource-augmentation approximation ratio for the test of Corollary 4.1 is described in the Theorem below.

Theorem 4.2 *Any real-time instance I that is* feasible *(according to some hypothetically optimal feasibility test) upon m-processors of unit capacity is guaranteed to satisfy the conditions of Equation 4.8 and 4.9 of Corollary 4.1 on an m-processor platform where each processor has speed $4 - \frac{1}{m}$.*

Proof: Let I be a real-time instance that is feasible on platform Π with m unit capacity processors. Consider I now scheduled upon platform $(4-\frac{1}{m})\cdot\Pi$ (i.e., platform

Π where each processor has been sped up by a factor of $(4 - \frac{1}{m})$. If we normalize the execution time of I to the speed of processor $(4 - \frac{1}{m}) \cdot \Pi$, we obtain a new representation of I (denoted by I') on the faster processing platform; for each job $J_i \in I$, there is a job $J_i' \in I'$ with identical arrival time and relative deadline, but with E_i' equal to $\frac{mE_i}{4m-1}$. By Equation 3.1, for all $0 \leq t_1 < t_2$, $\mathsf{demand}(I', t_1, t_2)$ is equal to $\frac{m}{4m-1} \cdot \mathsf{demand}(I, t_1, t_2)$; by Equations 3.3 and 3.2,

$$\begin{aligned} \mathsf{load}(I') &= \frac{m}{4m-1} \cdot \mathsf{load}(I) \\ \mathsf{max\text{-}job\text{-}density}(I') &= \frac{m}{4m-1} \cdot \mathsf{max\text{-}job\text{-}density}(I). \end{aligned}$$

Since I is feasible on Π then by Lemma 3.2, it must be that $\mathsf{load}(I) \leq m$ and $\mathsf{max\text{-}job\text{-}density}(I) \leq 1$. Substituting into the equations above, we obtain

$$\begin{aligned} \mathsf{load}(I') &\leq \frac{m^2}{4m-1} \\ \mathsf{max\text{-}job\text{-}density}(I') &= \frac{m}{4m-1}. \end{aligned}$$

Thus, I' (equivalently I) is feasible by Corollary 4.1 on $(4 - \frac{1}{m}) \cdot \Pi$. ■

4.1.2 Tightness of the Bound

As stated, Corollary 4.1 asserts that our algorithm exhibits behavior that is, in a certain sense, no more than a factor of approximately four off optimal behavior. In fact, if we restrict our attention only to real-time systems that are characterized exclusively by their load and max-job-density parameters, we can show that our algorithm is actually within a factor of $2\frac{2}{3}$ of optimal behavior. We do this by proving that a necessary condition for instance I to be feasible is in fact tighter than assumed above:

there are instances I with $\mathsf{max\text{-}job\text{-}density}(I) = \frac{2}{3} + \epsilon$ and $\mathsf{load}(I) = \frac{2m}{3} + \epsilon$ for any positive ϵ, that are not feasible on m unit-capacity processors. Consider the following real-time instance I consisting of four jobs (each job is represented by the three-tuple (A_i, E_i, D_i)):

$$I = \{J_1 = (0, \frac{4}{3}, 2); J_2 = J_3 = (1, \frac{2}{3}, 1); J_4 = (1, \frac{4}{3}, 2)\} \ .$$

It may be verified that $\mathsf{max\text{-}job\text{-}density}(I) = \frac{2}{3}$ and $\mathsf{load}(I) = \frac{4}{3}$. This instance is feasible upon two processors; however, increasing the execution requirement of any of the four jobs by any amount at all would render it infeasible. For any (even) number of processors m, similar instances can be constructed with $\mathsf{max\text{-}job\text{-}density}$ two-thirds and load equal to $\frac{2m}{3}$. Since no algorithm, not even an optimal one, is able to schedule such an instance it therefore follows that *our algorithm is worse than optimal by a factor of no more than* $\frac{2/3}{1/4} = \frac{8}{3}$.

4.2 Full-Migration Feasibility

The results of the previous section certainly apply to the full-migration feasibility of a real-time instance, in addition to restricted-migration feasibility (observe that restricted-migration schedules are also full-migration schedules in which jobs do not migrate). The results contained in this section improve upon these sufficient feasibility analysis conditions for full-migration scheduling, and introduce schedulability conditions based on load and $\mathsf{max\text{-}job\text{-}density}$ for EDF scheduling.

Let us begin by stating the main full-migration feasibility result that we have obtained. After some discussion of the result, we will formally prove it in Section 4.2.1.

Theorem 4.3 *For real-time instance I and identical multiprocessor platform Π comprised of m unit-speed processors, if*

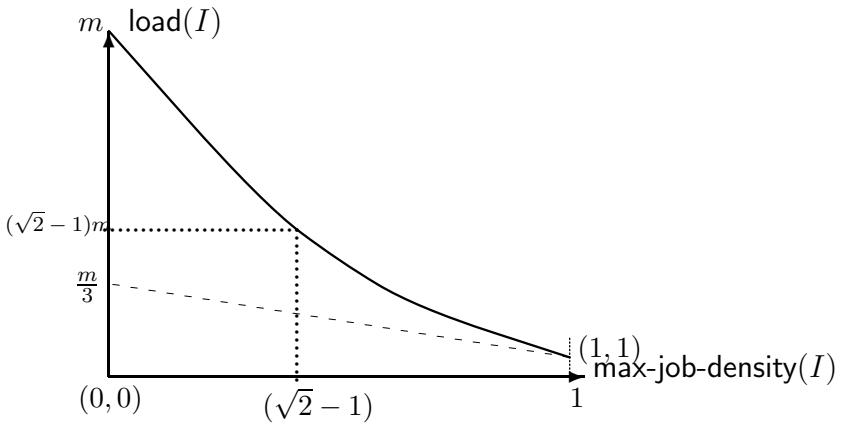


Figure 4.4: The feasibility region, as defined by Equation 4.10. All real-time instances I for which (max-job-density(I), load(I)) lies beneath the curved solid line are guaranteed feasible on m unit-capacity processors. The dashed line illustrates the restricted-migration feasibility bound of Theorem 4.1 (from previous section).

$$\mathsf{load}(I) < \frac{m - (m-2)\mathsf{max\text{-}job\text{-}density}(I)}{1 + \mathsf{max\text{-}job\text{-}density}(I)} \tag{4.10}$$

then I is globally feasible upon platform Π.

As in the previous section, our goal is to obtain sufficient conditions for preemptive multiprocessor feasibility (in this case under full-migration scheduling). Looking again at the feasibility region in the two-dimensional space $[0,1] \times [0,m]$, Theorem 4.3 yields the following feasibility region: for given m, any instance I satisfying Equation 4.10 is guaranteed to be feasible upon m unit-speed processors. The feasibility region for unit-capacity processors is depicted visually in Figure 4.4.

We may show that for any real-time instance I our feasibility conditions are within a constant factor of the optimal value of $\mathsf{load}(I)$ and $\mathsf{max\text{-}job\text{-}density}(I)$. Consider the following corollary to Theorem 4.3.

Corollary 4.2 *Any real-time instance I satisfying the following two conditions*

$$\mathsf{load}(I) \leq (\sqrt{2}-1)m \tag{4.11}$$

$$\mathsf{max\text{-}job\text{-}density}(I) \leq (\sqrt{2}-1) \tag{4.12}$$

is feasible upon an m-processor unit-capacity multiprocessor platform under full-migration multiprocessor scheduling.[2]

Proof: Assume that Conditions 4.11 and 4.12 hold for real-time instance I. Notice that the condition of Equation 4.10 of Theorem 4.3, decreases as $\mathsf{max\text{-}job\text{-}density}(I)$ increases. So, all instances I with $\mathsf{max\text{-}job\text{-}density}(I) \leq \sqrt{2} - 1$ and

$$\begin{aligned} & \mathsf{load}(I) < \tfrac{m-(m-2)(\sqrt{2}-1)}{1+(\sqrt{2}-1)} = (\sqrt{2}-1)m + 2 - \sqrt{2} \\ \Leftarrow \quad & \mathsf{load}(I) \leq (\sqrt{2}-1)m \end{aligned} \tag{4.13}$$

are feasible according to Theorem 4.3 on m unit-capacity processors. However, Equation 4.13 implies that all instances I that satisfy Conditions 4.11 and 4.12 are feasible on m unit-capacity processors. ■

As with Corollary 4.1, the result in Corollary 4.2 above can be considered to be an analog of a "utilization" test, or a "processor demand criteria" test, for uniprocessor systems. According to Lemma 3.2, a necessary condition for real-time instance to be feasible on m unit-capacity processors is that $\mathsf{load}(I) \leq m$ and $\mathsf{max\text{-}job\text{-}density}(I) \leq 1$; by Corollary 4.2 above, it is *sufficient* that $\mathsf{load}(I) \leq (\sqrt{2}-1)m$ and $\mathsf{max\text{-}job\text{-}density}(I) \leq \sqrt{2} - 1$. Hence, to within a constant factor of less than 2.5, we have obtained bounds on the values of $\mathsf{load}(I)$ and $\mathsf{max\text{-}job\text{-}density}(I)$ that are needed (and suffice) for feasibility. The following theorem states the resource augmentation guarantee of Corollary 4.2. The proof of the theorem is nearly identical to Theorem 4.2 and will be omitted.

Theorem 4.4 *Any real-time instance I that is* feasible *(according to some hypothetically optimal feasibility test) upon m-processors of unit capacity is guaranteed to satisfy the conditions of Equations 4.11 and 4.12 of Corollary 4.2 of Corollary 4.1 on*

[2]An alternative statement of this corollary could be: *the point* $\left(\sqrt{2}-1, (\sqrt{2}-1)m\right)$ *lies in the feasibility region of Figure 4.4* — this point is depicted by the dotted lines in the figure.

an m-processor platform where each processor has speed $\sqrt{2}+1$.

§ **How tight is this bound?.** Theorem 4.4 asserts that our algorithm exhibits behavior that is, in a certain sense, no more than a factor of approximately 2.5 off optimal behavior. In fact, if we restrict our attention only to real-time systems that are characterized exclusively by their load and max-job-density parameters, we can show that our algorithm is actually within a factor of approximately 1.61 of optimal behavior in this same sense. Consider the same real-time instance I with $\mathsf{load}(I) = \frac{2m}{3} + \epsilon$ and $\mathsf{max\text{-}job\text{-}density}(I) = \frac{2}{3} + \epsilon$ from Section 4.1.2; recall that I is infeasible on two processors of unit capacity for all $\epsilon > 0$; therefore, no algorithm (not even an optimal algorithm) would be able to declare this task system feasible. Comparing the load of this infeasible task system with the bounds of Corollary 4.2, it follows that *our algorithm is worse than optimal by a factor of no more than* $\frac{2/3}{\sqrt{2}-1} \approx 1.61$.

4.2.1 Proof of Theorem 4.3

In this section, we give the detailed proof of the feasibility conditions of Theorem 4.3. The proof will rely heavily on the definitions and notation of Sections 1.3.1 and 1.4.1. Section 4.2.1.1 gives some additional preliminary notation needed for the proof. Section 4.2.1.2 gives a detailed outline of the main steps of the proof. Section 4.2.1.3 contains the entire proof.

4.2.1.1 Notation

Throughout this section, we redefine the order and indexing of jobs to be in non-decreasing order of absolute deadlines (i.e., for all $J_i, J_j \in I$, $i < j$ if and only if $A_i + D_i \leq A_j + D_j$). Inevitably, in any schedule on a processing platform a job may be prevented from executing (even though it has remaining execution) because

the platform is busy executing other jobs. If job J_k is prevented from executing because job J_i was being executed on the platform, we say that J_i *interfered* with the execution of J_k. Below, we define one form of interference that will be used throughout the proof of Theorem 4.3: *arrival-time interference*. The next definition formally defines a predicate for arrival-time interference for any two jobs in schedule S_I for instance I.

Definition 4.1 (Arrival-Time Interference Predicate) *A job J_i* arrival-time interferes *with job J_k in a given schedule S_I if J_i arrives earlier than J_k (or at exactly the same time, but has lower index), and there exists a non-empty interval (t_1, t_2) in the intersection of $[A_i, A_i + D_i)$ and $[A_k, A_k + D_k)$ where for all $t \in (t_1, t_2)$:*

1. *J_i is executing.*
2. *No processor is idle.*
3. *No processor is executing J_k.*

Let $\phi : \mathbb{S}_I \times I \times I \to \{\text{true}, \text{false}\}$ be the predicate which denotes that J_i arrival-time interferes with J_k. Formally,

$$
\begin{aligned}
&\phi(S_I, J_i, J_k) \\
&\quad \stackrel{def}{=} \quad ((A_i < A_k) \vee ((A_i = A_k) \wedge (i < k))) \\
&\qquad \bigwedge \\
&\qquad [\exists (t_1, t_2) \subseteq [A_i, A_i + D_i) \cap [A_k, A_k + D_k) \\
&\qquad\quad \text{such that} \\
&\qquad\qquad (t_2 > t_1) \\
&\qquad \wedge \quad (\forall t \in (t_1, t_2) : \exists \pi_\ell \in \Pi :: S_I(\pi_\ell, t, J_i) = 1) \\
&\qquad \wedge \quad (\forall t \in (t_1, t_2), \pi_\ell \in \Pi : S_I(\pi_\ell, t) \neq \bot, J_k)].
\end{aligned}
\tag{4.14}
$$

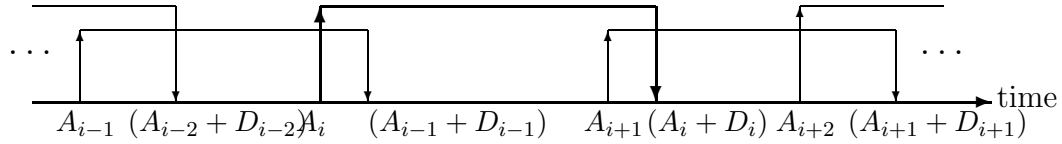


Figure 4.5: Visual depiction of a spanning chain.

The final definition in this notation section describes a subset of jobs whose arrival sequence forms a *chain*. The proof of Theorem 4.3 will reason about the amount of execution that must occur on the processing platform over a chain.

Definition 4.2 (Spanning Chain for finite instance I_{finite}) *Let* $I_{\text{finite}} \stackrel{def}{=} \{J_1, \ldots, J_n\}$*, ordered according to absolute deadline. Let* J_{ℓ_1} *be the job of* I_{finite} *with the earliest arrival time. A spanning chain for* I_{finite} *is a sequence of jobs* $J_{\ell_1}, J_{\ell_2}, \ldots, J_{\ell_r}$ *with the following three properties:*

1. $J_{\ell_r} = J_n$,
2. *for all* $1 < i \leq r$, $A_{\ell_{i-1}} < A_{\ell_i} \leq A_{\ell_{i-1}} + D_{\ell_{i-1}}$,
3. *for all* $1 < i < r$, $A_{\ell_{i-1}} + D_{\ell_{i-1}} < A_{\ell_{i+1}}$.

Figure 4.5 gives a visual illustration of a segment of a spanning chain. Notice, the spanning chain is equivalent to ensuring that every time point in the interval $[A_1, A_n + D_n)$ is contained in the scheduling window of at least one job of the spanning chain, but not more than two jobs.

4.2.1.2 Outline

The following is an informal outline of the steps taken in the proof of Theorem 4.3 (in the next subsection).

1. We will prove the contrapositive of Theorem 4.3. That is, if real-time instance I is infeasible, then $\mathsf{load}(I) \geq \frac{m-(m-2)\mathsf{max\text{-}job\text{-}density}(I)}{1+\mathsf{max\text{-}job\text{-}density}(I)}$. The first step of the proof is to assume that I is infeasible on m-processor platform Π.

2. We consider I_{infeas} which is a subset of real-time instance I. I_{infeas} is defined to be the first n jobs of I (in order of index) where $\{J_1, J_2, \ldots, J_{n-1}\}$ is feasible on Π, but $I_{\text{infeas}} \stackrel{\text{def}}{=} \{J_1, \ldots, J_{n-1}, J_n\}$ is infeasible on Π. (Recall, in this section, it is assumed that jobs are indexed by their absolute deadlines).

3. We define a schedule $S'_{I_{\text{infeas}}}$ (Equation 4.15 below) on Π in which jobs $J_1, \ldots, J_{n-1}$ meet their deadline, but J_n misses its deadline. We then consider the set of all possible sequences of jobs $J_{\ell_1}, J_{\ell_2}, \ldots, J_{\ell_s} \in I_{\text{infeas}}$ that arrival-time interfere with each other, and J_{ℓ_s} arrival-time interferes with J_n (i.e., for $1 \leq i < s$, $\phi(S'_{I_{\text{infeas}}}, J_{\ell_i}, J_{\ell_{i+1}})$ and $\phi(S'_{I_{\text{infeas}}}, J_{\ell_s}, J_n)$).

4. Over all possible sequences from the previous step, we consider the sequence $\gamma = \{J_{\gamma_1}, J_{\gamma_2}, \ldots, J_{\gamma_s}\}$ with the earliest arriving job. Let A_{γ_1} be the arrival time of the earliest job in γ. Let I' be the subset of jobs of I_{infeas} with arrival-times greater than A_{γ_1}. We will show that the work (with respect to I' and the system work function) done over the interval of each job J_i in $\gamma \cup \{J_n\}$ must be at least $\left[m - (m-1)\frac{E_i}{D_i}\right] D_i$. Lemma 4.1 proves this statement.

5. From the sequence $\gamma \cup \{J_n\}$, we argue that there must exist a spanning chain χ (according to Definition 4.2). Each job J_{χ_i} of the spanning chain also has the property that the amount of work done over its scheduling window must be at least $[m - (m-1)\mathsf{max\text{-}job\text{-}density}(I)]\, D_{\chi_i}$ (Corollary 4.3).

6. We use the lower bound on the amount of work from the previous step and properties of the spanning chain χ to show that the $\mathsf{load}(I')$ must be at least $\frac{m-(m-2)\mathsf{max\text{-}job\text{-}density}(I)}{1+\mathsf{max\text{-}job\text{-}density}(I)}$.

7. Since $I' \subseteq I$, $\mathsf{load}(I') \leq \mathsf{load}(I)$. Thus, if I is infeasible, the load must equal or exceed the bound from the previous step. Theorem 4.3 follows from the contrapositive of this statement.

4.2.1.3 Proof

§ **Construction of $S'_{I_{\text{infeas}}}$ and I'.** Assume the notation defined in Step 2 of the proof outline. That is, let Π be a unit-speed identical m-processor platform; let I_{infeas} be the first n jobs of I such that $\{J_1, J_2, \ldots, J_{n-1}\}$ is feasible on Π, but $I_{\text{infeas}} \stackrel{\text{def}}{=} \{J_1, \ldots, J_{n-1}, J_n\}$ is infeasible on Π.

We now describe how to construct the schedule $S'_{I_{\text{infeas}}}$ over the jobs of I_{infeas}. Informally, $S'_{I_{\text{infeas}}}$ is the schedule which maximizes the amount of time that J_n executes and jobs $I_{\text{infeas}} - \{J_n\}$ complete by their deadlines. Let σ be the set of all schedules for I_{infeas} on Π that are valid for $I_{\text{infeas}} - \{J_n\}$, and satisfy only Conditions 1 and 2 of validity (Definition 1.6), but not Condition 3 for J_n (i.e., job J_n does not complete execution by its deadline). Informally, σ is the set of schedules of I_{infeas} on Π such that J_n misses a deadline, but all other jobs meet their deadline. Note that σ is non-empty due to the fact that $I_{\text{infeas}} - \{J_n\}$ is feasible. We may now define schedule $S'_{I_{\text{infeas}}}$:

$$S'_{I_{\text{infeas}}} \stackrel{\text{def}}{=} \arg\max_{S \in \sigma} \{W(S, J_n, A_n, A_n + D_n)\}. \tag{4.15}$$

Let Γ be the set of all possible sequences of jobs of I_{infeas}, ending with J_n, such that each job in the sequence arrival-time interferes with the subsequent job (Γ may be empty). Formally,

$$\Gamma \stackrel{\text{def}}{=} \{\{J_{a_1}, J_{a_2}, \ldots, J_{a_s}\} \subseteq I_{\text{infeas}} | \phi(S'_{I_{\text{infeas}}}, J_{a_1}, J_{a_2}) \wedge \ldots \wedge \phi(S'_{I_{\text{infeas}}}, J_{a_s}, J_n)\}. \tag{4.16}$$

Define $\gamma \in \Gamma$ to be the *maximum length interference sequence* with the earliest arriving job (i.e., for all $\delta \in \Gamma$, $A_{\gamma_1} < A_{\delta_1}$ or $((A_{\gamma_1} = A_{\delta_1}) \wedge (\gamma_1 < \delta_1))$). We will now construct another real-time instance I' that is all jobs of I_{infeas} that arrive after A_{γ_1}– i.e., $I' \stackrel{\text{def}}{=} \{J_i \in I_{\text{infeas}} | A_i \geq A_{\gamma_1}\}$.

The next lemma gives a lower bound on the amount of work that must be done on behalf of jobs in I' over each job in the maximum interference sequence for schedule $S'_{I_{\text{infeas}}}$.

Lemma 4.1 *For all jobs J_{γ_i} in the maximum length interference sequence of schedule $S'_{I_{\text{infeas}}}$ plus job J_n (i.e., $\gamma \cup \{J_n\}$),*

$$W_{I'}(S'_{I_{\text{infeas}}}, A_{\gamma_i}, A_{\gamma_i} + D_{\gamma_i}) \geq D_{\gamma_i} \left[m - (m-1)\frac{E_{\gamma_i}}{D_{\gamma_i}} \right].$$

Proof: The proof is by contradiction; assume that there exists a $J_{\gamma_i} \in \gamma \cup \{J_n\}$ where

$$\begin{aligned} W_{I'}(S'_{I_{\text{infeas}}}, A_{\gamma_i}, A_{\gamma_i} + D_{\gamma_i}) &< D_{\gamma_i} \left[m - (m-1)\tfrac{E_{\gamma_i}}{D_{\gamma_i}} \right] \\ &= m(D_{\gamma_i} - E_{\gamma_i}) + E_{\gamma_i}. \end{aligned} \tag{4.17}$$

Assume that $E_{\gamma_i} < D_{\gamma_i}$; otherwise, Equation 4.17 is vacuously false. Let $E'_{\gamma_i} \leq E_{\gamma_i}$ be the amount of time during which J_{γ_i} executes in $S'_{I_{\text{infeas}}}$ over the interval $[A_{\gamma_i}, A_{\gamma_i} + D_{\gamma_i})$. Let Y_{γ_i} be the set of non-overlapping contiguous intervals during J_{γ_i}'s scheduling window in which no processor is executing J_{γ_i}, i.e., $Y_{\gamma_i} \stackrel{\text{def}}{=} \{(t_1, t_2) \subseteq [A_{\gamma_i}, A_{\gamma_i} + D_{\gamma_i})| \ (\forall \pi_k \in \Pi, t \in (t_1, t_2) : S'_{I_{\text{infeas}}}(\pi_k, t) \neq J_{\gamma_i}) \wedge (t_2 > t_1)\}$. Since J_{γ_i} can only execute for E'_{γ_i} amount of time, then the sum of the interval lengths of Y_{γ_i} is $\sum_{(t_1,t_2) \in Y_{\gamma_i}} (t_2 - t_1) = D_{\gamma_i} - E'_{\gamma_i} \geq D_{\gamma_i} - E_{\gamma_i}$. Note by Equation 4.17, there exists an interval time (t'_1, t'_2) that is a subset of an interval of Y_{γ_i} such that at least one processor is idle for all $t \in (t'_1, t'_2)$; this fact follows by observing that if such a (t'_1, t'_2) did not exist the amount of work $W_{I'-\{J_{\gamma_i}\}}(S'_{I_{\text{infeas}}}, A_{\gamma_i}, A_{\gamma_i} + D_{\gamma_i}) = m(D_{\gamma_i} - E'_{\gamma_i}) + E'_{\gamma_i} \geq$

$m(D_{\gamma_i} - E_{\gamma_i}) + E_{\gamma_i}$, contradicting Inequality 4.17. Let Z_{γ_i} be the set of all such idle sub-intervals of Y_{γ_i}; that is, $Z_{\gamma_i} \stackrel{\text{def}}{=} \{(t'_1, t'_2) \subseteq (t_1, t_2)\ (\in Y_{\gamma_i}) \mid (\forall t \in (t'_1, t'_2) : \exists \pi_k \in \Pi :: S'_{I_{\text{infeas}}}(\pi_k, t) = \perp) \wedge (t'_2 > t'_1)\}$. Note that Z_{γ_i} is non-empty.

If $J_{\gamma_i} = J_n$, the fact that there exists an idle point in an interval in Y_n contradicts the definition of $S'_{I_{\text{infeas}}}$; so, the lemma holds for J_n. We now consider $J_{\gamma_i} \neq J_n$ (note, if $J_{\gamma_i} = J_{\gamma_s}$ then $J_{\gamma_{i+1}}$ is J_n). From the definition of $\phi(S'_{I_{\text{infeas}}}, J_{\gamma_i}, J_{\gamma_{i+1}})$, there exists an $X_{\gamma_i} \stackrel{\text{def}}{=} (t_1, t_2) \subseteq [A_{\gamma_i}, A_{\gamma_i} + D_{\gamma_i}) \cap [A_{\gamma_{i+1}}, A_{\gamma_{i+1}} + D_{\gamma_{i+1}})$ where $|X_{\gamma_i}| > 0$ and for all $t \in X_{\gamma_i}$,

$$\left(\exists \pi_k \in \Pi, S'_{I_{\text{infeas}}}(\pi_k, t) = J_{\gamma_i}\right) \wedge \left(\forall \pi_k \in \Pi, S'_{I_{\text{infeas}}}(\pi_k, t) \neq J_{\gamma_{i+1}}\right). \tag{4.18}$$

Similarly, there exists $X_{\gamma_{i+1}} \subseteq \left([A_{\gamma_{i+1}}, A_{\gamma_{i+1}} + D_{\gamma_{i+1}}) \cap [A_{\gamma_{i+2}}, A_{\gamma_{i+2}} + D_{\gamma_{i+2}})\right), \ldots,$ $X_{\gamma_s} \subseteq ([A_{\gamma_s}, A_{\gamma_s} + D_{\gamma_s}) \cap [A_n, A_n + D_n))$. That is, for each job J_{γ_k} $(\gamma_k \geq \gamma_i)$ in the maximum length interference sequence there exists a well-defined interval X_{γ_k} in the intersection of J_{γ_k}'s scheduling window and its successor's $(J_{\gamma_{k+1}})$ scheduling window such that J_{γ_k} is executing while all other processors are busy, and $J_{\gamma_{k+1}}$ is not executing. Z_{γ_i} contains intervals during J_{γ_i}'s activation in which J_{γ_i} is not executing and there is an idle processor; Let (t_a, t_b) be any interval of set Z_{γ_i}. We may define a new schedule $S^{(0)}$ (based on $S'_{I_{\text{infeas}}}$) where we move $\min(|t_b - t_a|, |X_{\gamma_i}|)$ units of J_{γ_i}'s execution from times in X_{γ_i} to Z_{γ_i}. In doing this "swap," we have not violated any of the conditions of a valid schedule (Definition 1.6). Now in $S^{(0)}$ there exists $t \in X_{\gamma_i}$ and $\pi_k \in \Pi$ such that $S^{(0)}(\pi_k, t) = \perp$. Thus, we can move $\min(|t_b - t_a|, |X_{\gamma_i}|, |X_{\gamma_{i+1}}|)$ units of $J_{\gamma_{i+1}}$ execution from $X_{\gamma_{i+1}}$ to X_{γ_i}; define such a schedule to $S^{(1)}$ (based on schedule $S^{(0)}$). We can repeat this "swapping" procedure until we define $S^{(s-i+1)}$ where we allow $\min\left(|t_b - t_a|, |X_{\gamma_i}|, \ldots, |X_{\gamma_s}|, x_{\text{remaining}}\right)$ units of additional execution for J_n to occur in X_{γ_s} where $x_{\text{remaining}}$ is $\left(E_n - W(S'_{I_{\text{infeas}}}, J_n, A_n, A_n + D_n)\right)$. Thus, we have defined a schedule which is valid for $I_{\text{infeas}} - \{J_n\}$ (in any of our swappings

we have not violated Properties 1 through 3), and has

$$W(S^{(s-i+1)}, J_n, A_n, A_n + D_n) > W(S'_{I_{\text{infeas}}}, J_n, A_n, A_n + D_n).$$

The above inequality directly contradicts the definition of $S'_{I_{\text{infeas}}}$ in Equation 4.15; therefore, our supposition is false, and the lemma is true. ■

§ **Existence of a spanning chain for I'.** The maximum-length interference sequence is not necessarily guaranteed to be a spanning chain according to Definition 4.2. However, it may be shown that there must exist $\chi \subseteq \gamma \cup \{J_n\}$ that is a spanning chain. Let $\mathcal{C}$ be the set of all subsets of $\gamma \cup \{J_n\}$ that "cover" the interval $[A_{\gamma_1}, A_n + D_n)$. (A set X of sub-intervals covers the interval $[A_{\gamma_1}, A_n + D_n)$ if for each $t \in [A_{\gamma_1}, A_n + D_n)$ there is a sub-interval in X that contains the point t). Define, the minimum cover of interval $[A_{\gamma_1}, A_n + D_n)$ to be

$$\chi \stackrel{\text{def}}{=} \arg\min_{X \in \mathcal{C}} \{|X|\}. \tag{4.19}$$

It is relatively easy to see that for all times t in $[A_{\gamma_1}, A_n + D_n)$, at most two jobs in χ contain t in their activation; if not, then consider the job with the earliest arrival,J_{first} and the job with the latest deadline, J_{last} that cover t. By definition, all other jobs that contain t are contained in the union of the scheduling windows of J_{first} and J_{last}, and thus are not contained in χ. The fact that t is contained in the scheduling window of at most two jobs and at least one job of χ implies that χ is a spanning chain.

The existence of such a spanning chain is significant because we may use Lemma 4.1 to infer the amount of work that must be done over each job of the spanning tree. The next corollary describes the amount of work that must be done over each job of the spanning chain χ.

Corollary 4.3 *For each job* $J_{\chi_i} \in \chi$,

$$W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_i}, A_{\chi_i} + D_{\chi_i}) \geq D_{\chi_i} \left[m - (m-1)\textsf{max-job-density}(I)\right]. \tag{4.20}$$

§ **Lower bound on the load of I'.** In the remainder of this section, we complete our proof of Theorem 4.3 by deriving a lower bound on the load of I'. We obtain such a lower bound on I' by deriving a lower bound on the minimum amount of work (with respect to jobs of I') that can occur over the spanning chain χ (given by the set $\{J_{\chi_1}, \ldots, J_{\chi_r}\}$ ordered according to arrival-time). The derivation of the lower bound (Lemma 4.3) for χ is obtained by reasoning about a linear program $\mathcal{L}$ (Figure 4.6) that is indirectly related to I' and χ. $\mathcal{L}$ is significant because we may show (Lemma 4.2) that there exists a feasible[3] solution to $\mathcal{L}$ which corresponds to I' and $S'_{I_{\text{infeas}}}$. This solution has an objective value equal to $W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_1}, A_{\chi_r} + D_{\chi_r})$ (or, equivalently, $W_{I'}(S'_{I_{\text{infeas}}}, A_{\gamma_1}, A_n + D_n)$). Therefore, we may conclude that

$$W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_1}, A_{\chi_r} + D_{\chi_r}) \geq \text{ Optimal value of } \mathcal{L}. \tag{4.21}$$

We first note that throughout this section we will assume that $r > 1$. If $r = 1$, then $\chi = \{J_n\}$. However, we know from Lemma 4.1 that the work done over $[A_n, A_n + D_n)$ exceeds or equals $D_n \left[m - (m-1)\textsf{max-job-density}(I)\right]$. This trivially satisfies the lower bound of

$$D_n \left[\frac{m - (m-2)\textsf{max-job-density}(I)}{1 + \textsf{max-job-density}(I)}\right],$$

which will be provided by Lemma 4.3. Therefore, we consider only the non-trivial

[3] *Feasible* here refers to a non-negative assignment to the variables of $\mathcal{L}$ that satisfies each of the constraints in the system

Linear Program $\mathcal{L}$

Let $\vec{a}, \vec{b} \in \mathbb{R}^r_{(>0)}$ and $\vec{c}, \vec{d} \in \mathbb{R}^{r-1}_{(>0)}$.
Minimize

$$F(\vec{a}, \vec{b}, \vec{c}, \vec{d}) \stackrel{\text{def}}{=} m \left[\sum_{i=1}^{r} b_{\chi_i} + \sum_{i=1}^{r-1} d_{\chi_i} \right] + \sum_{i=1}^{r} a_{\chi_i} + \sum_{i=1}^{r-1} c_{\chi_i} \tag{4.22}$$

subject to the following constraints:

$$\begin{aligned} \textstyle\sum_{i=1}^{r} [a_{\chi_i} + b_{\chi_i}] \; & + \textstyle\sum_{i=1}^{r-1} [c_{\chi_i} + d_{\chi_i}] \\ & = A_{\chi_r} + D_{\chi_r} - A_{\chi_1} \quad (4.23a) \end{aligned}$$

$$\begin{aligned} \mathsf{max\text{-}job\text{-}density}(I)\,(b_{\chi_1} + d_{\chi_1}) \\ -\,(1 - \mathsf{max\text{-}job\text{-}density}(I))(a_{\chi_1} + c_{\chi_1}) \; & \geq 0 \quad (4.23b) \end{aligned}$$

$$\begin{aligned} \mathsf{max\text{-}job\text{-}density}(I)\left(d_{\chi_{i-1}} + b_{\chi_i} + d_{\chi_i}\right) \\ -\left(1 - \mathsf{max\text{-}job\text{-}density}(I)\right)(c_{\chi_{i-1}} + a_{\chi_i} + c_{\chi_i}) \; & \geq 0 \\ (\forall i : 1 < i < r-1) \quad & (4.23c) \end{aligned}$$

$$\begin{aligned} \mathsf{max\text{-}job\text{-}density}(I)\left(b_{\chi_r} + d_{\chi_{r-1}}\right) \\ -\left(1 - \mathsf{max\text{-}job\text{-}density}(I)\right)(a_{\chi_r} + c_{\chi_{r-1}}) \; & \geq 0. \quad (4.23d) \end{aligned}$$

Figure 4.6: Linear System representing the minimum amount of work done over the spanning chain χ (with respect to jobs of I').

case where $r > 1$ in the remainder of this section.

We will now informally describe $\mathcal{L}$. Over the interval $[A_{\chi_1}, A_{\chi_r} + D_{\chi_r})$, at least one processor must be busy executing a job of I'; otherwise, using similar reasoning to Lemma 4.1, we could obtain a contradiction to Equation 4.15. $\mathcal{L}$ formally describes an abstract system containing jobs of χ where the processors of platform Π over the interval $[A_{\chi_1}, A_{\chi_r} + D_{\chi_r})$ are either all busy or at least one processor is busy. Note that $\mathcal{L}$ does not necessarily correspond to a system physically obtainable from I_{infeas}; however, Equation 4.21 shows that an optimal solution to abstract system $\mathcal{L}$ can

provide a lower bound on the work done in system I_{infeas}. Informally, each variable of the linear system $\mathcal{L}$ has the following interpretation.

- a_{χ_i} represents the amount of time during the interval in which J_{χ_i} is the only active job of χ and at least one processor of Π is busy. $\vec{a} \stackrel{\text{def}}{=} (a_{\chi_1}, \ldots, a_{\chi_r})$.
- b_{χ_i} represents the amount of time during the interval in which J_{χ_i} is the only active job of χ, and all processors of Π are busy executing. $\vec{b} \stackrel{\text{def}}{=} (b_{\chi_1}, \ldots, b_{\chi_r})$.
- c_{χ_i} represents the amount of time during the interval in which J_{χ_i} and $J_{\chi_{i+1}}$ overlap (where $i < r$), and at least one processor of Π is busy. $\vec{c} \stackrel{\text{def}}{=} (c_{\chi_1}, \ldots, c_{\chi_{r-1}})$.
- d_{χ_i} represents the amount of time during the interval in which J_{χ_i} and $J_{\chi_{i+1}}$ overlap (where $i < r$), and all processors of Π are busy executing. $\vec{d} \stackrel{\text{def}}{=} (d_{\chi_1}, \ldots, d_{\chi_{r-1}})$.

The objective function, $F(\vec{a}, \vec{b}, \vec{c}, \vec{d})$, of system $\mathcal{L}$ (Equation 4.22) represents the minimum amount of work done by platform Π over the interval $[A_{\chi_1}, A_{\chi_r} + D_{\chi_r})$. The equality constraint (Equation 4.23a) specifies that the total of all the interval lengths represented by vectors $\vec{a}$, $\vec{b}$, $\vec{c}$, and $\vec{d}$ must sum to the length of interval $[A_{\chi_1}, A_{\chi_r} + D_{\chi_r})$. Inequality constraints (Equations 4.23b-d) enforce the lower-bound work requirements described by Corollary 4.3. To see that each constraint corresponds to the lower bound of Corollary 4.3, consider the constraint of Equation 4.23b:

$$
\begin{aligned}
& \mathsf{max\text{-}job\text{-}density}(I)(b_{\chi_1} + d_{\chi_1}) - (1 - \mathsf{max\text{-}job\text{-}density})(a_{\chi_1} + c_{\chi_1}) \geq 0 \\
\equiv\ & \mathsf{max\text{-}job\text{-}density}(I)(a_{\chi_1} + b_{\chi_1} + c_{\chi_1} + d_{\chi_1}) - (a_{\chi_1} + c_{\chi_1}) \geq 0 \\
\equiv\ & \mathsf{max\text{-}job\text{-}density}(I) D_{\chi_1} \geq a_{\chi_1} + c_{\chi_1}.
\end{aligned}
$$

$$
\Leftarrow \frac{E_{\chi_1}}{D_{\chi_1}} \cdot D_{\chi_1} \geq a_{\chi_1} + c_{\chi_1} \equiv E_{\chi_1} \geq a_{\chi_1} + c_{\chi_1}
$$

The last statement is implied by Lemma 4.1 since if $E_{\chi_1} < a_{\chi_1} + c_{\chi_1}$, then the total amount of time over $[A_{\chi_1}, A_{\chi_1} + D_{\chi_1})$ during which all processors are busy does not exceed $D_{\chi_1} - E_{\chi_1}$. By the arguments of Lemma 4.1 this would imply that we could execute J_n for more than schedule $S'_{I_{\text{infeas}}}$ which is a contradiction. We may justify the constraints of Equation 4.23(c-d) by similar arguments.

The next lemma shows the existence of a solution that corresponds to I', χ, and $S'_{I_{\text{infeas}}}$. Thus, the optimal value of $\mathcal{L}$ provides a lower bound on $W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_1}, A_{\chi_r} + D_{\chi_r})$.

Lemma 4.2 *There exists a feasible assigment to $\vec{a}$, $\vec{b}$, $\vec{c}$, and $\vec{d}$ in $\mathcal{L}$ such that the objective function value, $F(\vec{a}, \vec{b}, \vec{c}, \vec{d})$ of this solution is equal to $W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_1}, A_{\chi_r} + D_{\chi_r})$.*

Proof: Consider the following assignment to variables of $\mathcal{L}$:

$$a_{\chi_1} \stackrel{\text{def}}{=} [A_{\chi_2} - A_{\chi_1}] - \frac{W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_1}, A_{\chi_2}) - [A_{\chi_2} - A_{\chi_1}]}{m-1}$$

$$b_{\chi_1} \stackrel{\text{def}}{=} \frac{W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_1}, A_{\chi_2}) - [A_{\chi_2} - A_{\chi_1}]}{m-1}$$

$$(\forall i : 1 \leq i < r) :$$

$$c_{\chi_i} \stackrel{\text{def}}{=} [A_{\chi_i} + D_{\chi_i} - A_{\chi_{i+1}}] - \frac{W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_{i+1}}, A_{\chi_i} + D_{\chi_i}) - [A_{\chi_i} + D_{\chi_i} - A_{\chi_{i+1}}]}{m-1}$$

$$d_{\chi_i} \stackrel{\text{def}}{=} \frac{W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_{i+1}}, A_{\chi_i} + D_{\chi_i}) - [A_{\chi_i} + D_{\chi_i} - A_{\chi_{i+1}}]}{m-1}$$

$$(\forall i : 1 < i < r) :$$

$$a_{\chi_i} \stackrel{\text{def}}{=} [A_{\chi_i+1} - A_{\chi_{i-1}} - D_{\chi_{i-1}}] - \frac{W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_{i-1}}+D_{\chi_{i-1}}, A_{\chi_{i+1}}) - [A_{\chi_i+1} - A_{\chi_{i-1}} - D_{\chi_{i-1}}]}{m-1}$$

$$b_{\chi_i} \stackrel{\text{def}}{=} \frac{W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_{i-1}}+D_{\chi_{i-1}}, A_{\chi_{i+1}}) - [A_{\chi_i+1} - A_{\chi_{i-1}} - D_{\chi_{i-1}}]}{m-1}.$$

$$a_{\chi_r} \stackrel{\text{def}}{=} [A_{\chi_r} + D_{\chi_r} - A_{\chi_{r-1}} - D_{\chi_{r-1}}] - \frac{W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_{r-1}}+D_{\chi_{r-1}}, A_{\chi_r}+D_{\chi_r}) - [A_{\chi_r}+D_{\chi_r} - A_{\chi_{r-1}} - D_{\chi_{r-1}}]}{m-1}$$

$$b_{\chi_r} \stackrel{\text{def}}{=} \frac{W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_{r-1}}+D_{\chi_{r-1}}, A_{\chi_r}+D_{\chi_r}) - [A_{\chi_r}+D_{\chi_r} - A_{\chi_{r-1}} - D_{\chi_{r-1}}]}{m-1}$$

It is a relatively straightforward exercise (through algebra and an application of Corollary 4.3) to show that the above assignment satisfies the constraints of Equations 4.23(a-d), and that $F(\vec{a}, \vec{b}, \vec{c}, \vec{d}) = W_{I'}(S'_{I_{\text{infeas}}}, A_{\chi_1}, A_{\chi_r} + D_{\chi_r})$. ■

The final lemma of this section gives the optimal value of $\mathcal{L}$ which will be ultimately used in the lower bound on $\mathsf{load}(I)$.

Lemma 4.3 *The optimal objective function value of $\mathcal{L}$ is*

$$[A_{\chi_r} + D_{\chi_r} - A_{\chi_1}] \left(\frac{m - (m-2)\mathsf{max\text{-}job\text{-}density}(I)}{1 + \mathsf{max\text{-}job\text{-}density}(I)} \right) \tag{4.23}$$

Proof: Consider the following solution to $\mathcal{L}$:

$$\begin{aligned}
a_{\chi_1} &= \frac{\mathsf{max\text{-}job\text{-}density}(I)}{1+\mathsf{max\text{-}job\text{-}density}(I)} \left[\frac{A_{\chi_r}+D_{\chi_r}-A_{\chi_1}}{r-1} \right] \\
a_{\chi_i} &= \frac{2\mathsf{max\text{-}job\text{-}density}(I)}{1+\mathsf{max\text{-}job\text{-}density}(I)} \left[\frac{A_{\chi_r}+D_{\chi_r}-A_{\chi_1}}{r-1} \right], \quad \forall i = 2, \ldots, r-1 \\
a_{\chi_r} &= \frac{\mathsf{max\text{-}job\text{-}density}(I)}{1+\mathsf{max\text{-}job\text{-}density}(I)} \left[\frac{A_{\chi_r}+D_{\chi_r}-A_{\chi_1}}{r-1} \right] \\
b_{\chi_i} &= 0, \qquad \forall i = 1, \ldots, r \\
c_{\chi_i} &= 0, \qquad \forall i = 1, \ldots, r-1 \\
d_{\chi_i} &= \frac{1-\mathsf{max\text{-}job\text{-}density}(I)}{1+\mathsf{max\text{-}job\text{-}density}(I)} \left[\frac{A_{\chi_r}+D_{\chi_r}-A_{\chi_1}}{r-1} \right] \quad \forall i = 1, \ldots, r-1
\end{aligned}$$

It can easily be verified that is a feasible solution to $\mathcal{L}$ with objective function value $F(a, b, c, d)$ equal to the value in Equation 4.23.

We may obtain the dual linear program $\overline{\mathcal{L}}$ by mechanical transformation from $\mathcal{L}$. The dual program $\overline{\mathcal{L}}$ is shown in Figure 4.7.

For the dual program $\overline{\mathcal{L}}$, it may also be verified that

$$\begin{aligned} \omega_0 &= \frac{m\big(1-\textsf{max-job-density}(I)\big)+2\textsf{max-job-density}(I)}{1+\textsf{max-job-density}(I)} \\ \omega_i &= \frac{m-1}{1+\textsf{max-job-density}(I)} \qquad\qquad \forall i = 1, \ldots, r \end{aligned}$$

is a feasible solution to $\overline{\mathcal{L}}$ with objective function value $G(\omega_0, \omega_1, \ldots, \omega_r)$ equal to the value in Equation 4.23. It is known (e.g., see (Murty, 1983), Theorem 4.4) that if there exist solutions to both the primal and dual linear program with the same objective value z, then z is an optimal solution in both. Therefore, Equation 4.23 is an optimal solution for $\mathcal{L}$. ■

Dual Linear Program $\overline{\mathcal{L}}$

Let $\omega_0 \in \mathbb{R}^+$ and $\omega_1, \ldots, \omega_r \in \mathbb{R}$
Maximize

$$G(\omega_0, \omega_1, \omega_2, \ldots, \omega_r) \stackrel{\text{def}}{=} [A_{\chi_r} + A_{\chi_r} - A_{\chi_1}] \times \omega_0 \tag{4.24}$$

subject to the following constraints:

$$\begin{aligned} &\omega_0 - (1 - \textsf{max-job-density}(I))\omega_i \leq 1 && \text{for all } i = 1, \ldots, r && (4.25a) \\ &\omega_0 + \textsf{max-job-density}(I)\omega_i \leq m && \text{for all } i = 1, \ldots, r && (4.25b) \\ &\omega_0 - (1 - \textsf{max-job-density}(I))(\omega_i + \omega_{i+1}) \leq 1 && \text{for all } i = 1, \ldots, r-1 && (4.25c) \\ &\omega_0 + \textsf{max-job-density}(I)(\omega_i + \omega_{i+1}) \leq m && \text{for all } i = 1, \ldots, r-1 && (4.25d) \end{aligned} \tag{4.25}$$

Figure 4.7: The dual linear program to $\mathcal{L}$.

We may now finally complete the proof of Theorem 4.3:

Proof of Theorem 4.3

By Equation 4.21 and Lemma 4.3, the minimum amount of work done over the interval $[A_{\chi_1}, A_{\chi_r} + D_{\chi_r})$ is

$$[A_{\chi_r} + D_{\chi_r} - A_{\chi_1}] \left(\frac{m - (m-2)\mathsf{max\text{-}job\text{-}density}(I)}{1 + \mathsf{max\text{-}job\text{-}density}(I)} \right).$$

A lower bound on the $\mathsf{load}(I')$ is obtained by dividing the above expression by $[A_{\chi_r} + D_{\chi_r} - A_{\chi_1}]$. This immediately implies

$$\frac{m - (m-2)\mathsf{max\text{-}job\text{-}density}(I)}{1 + \mathsf{max\text{-}job\text{-}density}(I)} \leq \mathsf{load}(I') \leq \mathsf{load}(I) \tag{4.26}$$

The last inequality follows because $I' \subseteq I_{\text{infeas}} \subseteq I$.

By Step 7 of the proof outline, we take the contrapositive of this statement to obtain the theorem.

■

4.3 Summary

Feasibility on preemptive uniprocessors is well understood; in the notation of this dissertation, a necessary and sufficient condition for any real-time instance I to be feasible (and EDF schedulable) upon a unit-capacity uniprocessor is that

$$\mathsf{load}(I) \leq 1 \text{ and } \mathsf{max\text{-}job\text{-}density}(I) \leq 1 \ .$$

In Chapter 3 of this dissertation, we have already shown necessary conditions for feasibility on preemptive multiprocessors (Lemma 3.2). In this chapter, we obtained (Theorems 4.1 and 4.3) sufficient conditions for a real-time instance I, characterized only by its load and max-job-density parameters, to be feasible on a multiprocessor

platform (in both the restricted- and full-migration setting). Since we have proven feasibility for the general real-time workload model of real-time instances, the results obtained in this chapter are applicable to any task model where load and max-job-density may be obtained (see Chapter 3). Furthermore, the feasibility tests obtained in this chapter are at most a small constant factor from the optimal feasibility tests (Corollaries 4.1 and 4.2), in terms of resource augmentation.

Chapter 5

The Impossibility of Optimal Online Multiprocessor Scheduling Algorithms for General Task Systems

After reading the previous chapter on feasibility, it is natural to wonder: *does there exist an algorithm which is guaranteed to successfully schedule any feasible general task system on a multiprocessor platform*? In other words, does there exist optimal scheduling algorithms for general task models? For LL task systems, the answer to that question is "yes." (Srinivasan and Anderson, 2002) describe a Pfair-based algorithm that is optimal for LL and periodic task systems. For the most general real-time workload abstraction, arbitrary real-time instances, (Dertouzos and Mok, 1989) show that optimal online scheduling of arbitrary real-time instances that are not known *a priori* is impossible. Thus, optimality exists for some of the stricter real-time task models and does not exist for the most general model of real-time work. This immediately prompts another question: for which general real-time recurrent

task models do optimal scheduling algorithms exist?

Unfortunately, for all the general task models discussed in this dissertation, optimal online scheduling is, in fact, impossible. In this chapter, we show that optimal online scheduling of sporadic task systems is impossible. This immediately implies that optimal online scheduling of any task model that generalizes the sporadic task system 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.

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 τ on some processing platform Π.

2. Prove that the task system is feasible a multiprocessor platform Π. This means that for any real-time instance generated by τ on Π there exists a schedule on Π that will meet all deadlines.

3. For the feasible task system τ, show there exists a set of real-time instances generated by τ 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 active jobs to schedule (i.e., there are more active 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 chapter, we give the details of Steps 1 and 3 which are contained in the next section. 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 chapter; therefore, we have decided to defer the details of Step 2 until Appendix A.

5.1 Impossibility of Optimal Online Scheduling

In accordance with Step 1 of the above approach, consider the following task system, τ_{example}, comprised of six tasks (recall the a sporadic task is specified by three-tuple (e_i, d_i, p_i)).

$$\begin{aligned} &\bullet\ \tau_1 = (2, 2, 5) \\ &\bullet\ \tau_2 = (1, 1, 5) \\ &\bullet\ \tau_3 = (1, 2, 6) \\ &\bullet\ \tau_4 = (2, 4, 100) \\ &\bullet\ \tau_5 = (2, 6, 100) \\ &\bullet\ \tau_6 = (4, 8, 100) \end{aligned} \tag{5.1}$$

Theorem 5.1 *τ_{example} is feasible on two processors.*

Proof: Proved in Appendix A.3. ■

Lemma 5.1 *No non-clairvoyant, optimal online algorithm exists for the multiprocessor scheduling real-time sporadic task systems on two processors.*

Proof: The proof is by contradiction. Assume there exists an optimal online algorithm, $\mathcal{A}$, for scheduling sporadic real-time tasks on two processors. Then, by Theorem 5.1, $\mathcal{A}$ must find a valid schedule for τ_{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 1.6). Figure 5.1a shows task system τ_{example}.

Let each task of τ_{example} release a job at time zero. Figure 5.1b shows the slots at which $\mathcal{A}$ must execute τ_1, τ_2, τ_3, and τ_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 τ_{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 τ_5.

1. $\mathcal{A}$ schedules τ_5 for x $(0 < x \leq 2)$ units of time in the interval $(2, 4]$.

2. $\mathcal{A}$ does not schedule τ_5 in the interval $(2, 4]$.

Since $\mathcal{A}$ is an online scheduling algorithm, by Definition 1.3, 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 τ_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 τ_1, τ_2, and τ_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 τ_1, τ_2,

and τ_3 over the interval $(6, 8]$; therefore, τ_6 may not execute over the interval $(6, 8]$ (otherwise, either τ_1, τ_2, or τ_3 will miss a deadline. This implies that τ_6 must execute in the interval $(2, 6]$ given real-time instance I. However, $\mathcal{I}$ chose to execute τ_5 in $(2, 4]$ for x time units, and τ_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 τ_6 may execute in the interval $(2, 4]$; τ_6 will miss a deadline at $t = 8$. Figure 5.2a shows this scenario.

2. *$\mathcal{A}$ does not schedule τ_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 τ_1 and τ_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 τ_1 and τ_2 over interval $(5, 6]$. However, since $\mathcal{A}$ chose not to execute τ_5 in the interval $(2, 4]$, τ_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 τ_1, τ_2, or τ_5 will miss a deadline in the interval $(5, 6]$. Figure 5.2b 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 5.2 *No non-clairvoyant, optimal online algorithm exists for the multipro-*

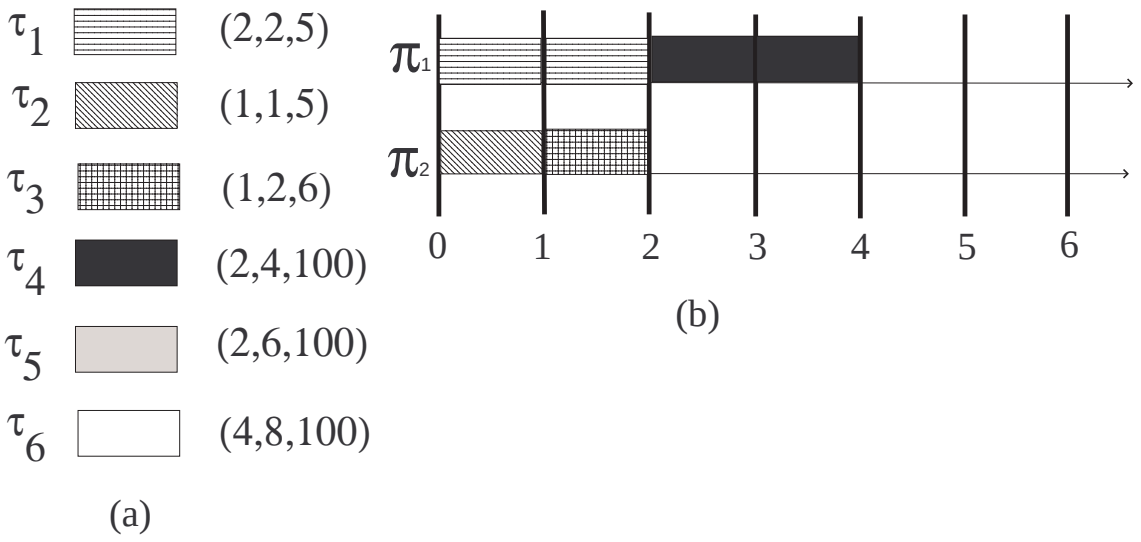


Figure 5.1: **(a)** Task system τ_{example}. **(b)** The times at which tasks τ_1, τ_2, τ_3, and τ_4 must execute.

cessor scheduling real-time sporadic task systems on two or more processors.

Proof: For any Π comprised of $m > 1$ identical unit-speed processors, consider the task system $\tau'_{\text{example}} \stackrel{\text{def}}{=} \tau_{\text{example}} \cup \{\tau'_1, \tau'_2, \ldots, \tau'_{m-2}\}$ where $\tau'_i = (1, 1, 1)$ for all $0 < i \leq m - 2$. It is easy to see that τ'_{example} is feasible on Π, as we can dedicate a processor to each of the tasks in $\{\tau'_1, \tau'_2, \ldots, \tau'_{m-2}\}$ and by Theorem 5.1 τ_{example} is feasible on the remaining two processors. The argument of Lemma 5.1 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 τ_{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 $\tau'^{\text{M}}_{\text{example}}$ specified in model M such that $I \in \mathcal{I}^{\text{M}}(\tau'^{\text{M}}_{\text{example}})$ if and only if $I \in \mathcal{I}^{\text{S}}(\tau'_{\text{example}})$. Therefore, the argument of Lemma 5.1 is unchanged for this more general task system.

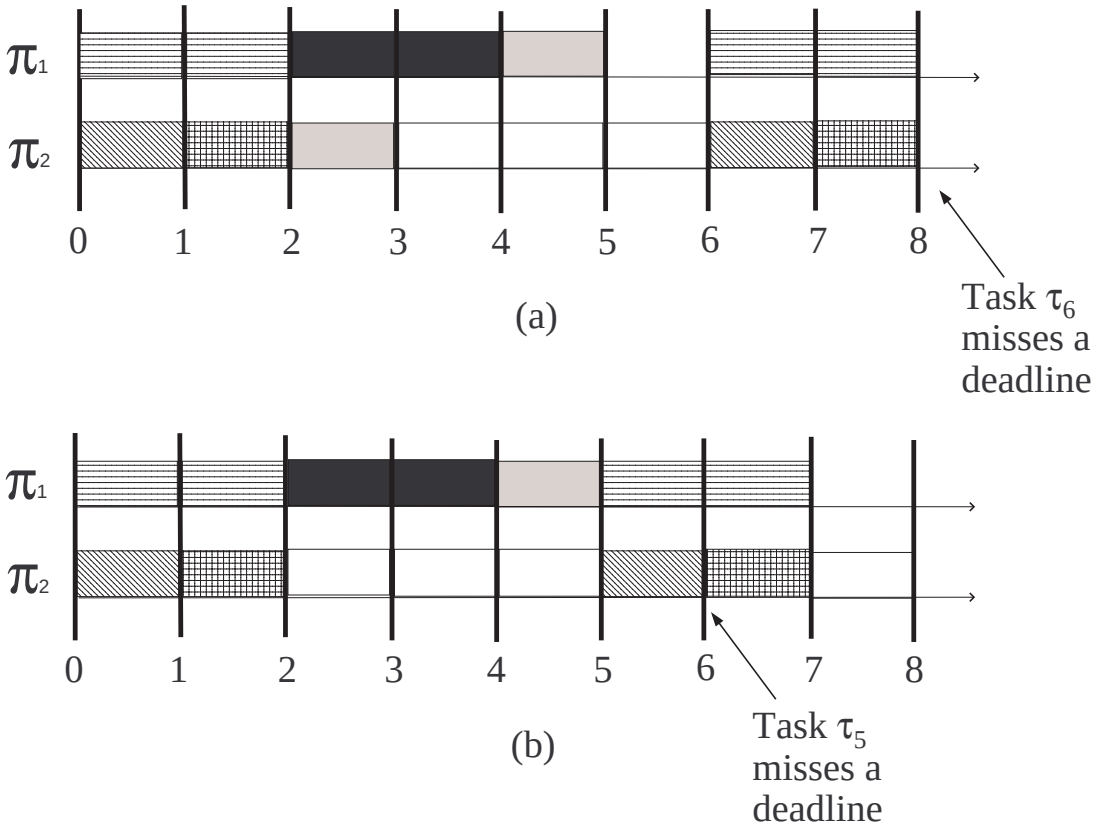


Figure 5.2: Scenario **(a)**: $\mathcal{A}$ schedules τ_5 for x $(0 < x \leq 2)$ units of time in the interval $(2, 4]$. Scenario **(b)**: $\mathcal{A}$ does not schedule τ_5 in the interval $(2, 4]$.

Corollary 5.1 *No non-clairvoyant, optimal online algorithm exists on two or more processors for the multiprocessor scheduling real-time task system in models that generalize the sporadic task model.*

5.2 Summary

In this chapter, 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 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. This chapter identified the

feasible task system and proved that no online scheduling algorithm can successfully schedule all feasible instances. Appendix A includes a detailed proof that this task system is feasible.

The consequence of this negative result are far-reaching in that algorithms that are optimal for LL and periodic task systems no longer retain their optimality for small generalizations of the 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 following chapters will explore using resource-augmentation as an analysis technique for identifying online scheduling algorithms with constant-factor approximation ratios.

Chapter 6

The Full-Migration Schedulability Analysis for General Task Systems

The impossibility result of the previous chapter implies that scheduling algorithms such as Pfair are no longer optimal for sporadic or more general task models. From Chapter 2, we saw that many other traditional online full- and restricted-migration algorithms suffer from Dhall's effect, implying these algorithms are provably non-optimal even for the simple LL task model. Taken together, these negative results may impart a rather pessimistic impression on the reader, concerning the efficient multiprocessor online scheduling of task systems in the sporadic and more general task models. However, we will see in this chapter, even though optimality is impossible, online scheduling and schedulability analysis of general task systems with constant-factor approximation ratios(in terms of resource augmentation) is achievable. In fact, many traditional online scheduling algorithms such as EDF or DM have schedulability tests with constant-factor resource-augmentation approximation ratios.

The remainder of this chapter is organized as follows. In Section 6.1, we introduce some additional notation useful in reasoning about the fixed job-priority (and fixed task-priority) scheduling of real-time instances. In Section 6.2, we obtain suf-

ficient conditions for the multiprocessor DM-schedulability of real-time instances. In Section 6.3, we derive sufficient conditions for EDF-schedulability. Both Sections 6.2 and 6.3 present resource augmentation bounds for their respective schedulability tests.

6.1 Notation

For each job J_i in real-time instance I, recall from Section 1.3 of Chapter 1 that we use the parameter $\rho(J_i)$ to denote the priority assigned to J_i by a fixed-job-priority scheduling algorithm. Given any priority-level p, we will now overload the definition of demand to include the execution requirements of jobs in the instance *with priority* $\geq p$, that have both their arrival times and their deadlines within the interval:

$$\mathsf{demand}(p, I, t_1, t_2) \stackrel{\text{def}}{=} \sum_{(J_i \in I) \wedge (\rho(J_i) \geq p) \wedge (t_1 \leq A_i) \wedge (A_i + D_i \leq t_2)} E_i.$$

Similarly, load may be overloaded with respect to priority-level p; $\mathsf{load}(p, I)$ as follows:

$$\mathsf{load}(p, I) \stackrel{\text{def}}{=} \max_{t_1 < t_2} \frac{\mathsf{demand}(p, I, t_1, t_2)}{t_2 - t_1}. \tag{6.1}$$

Intuitively, $\mathsf{load}(p, I)$ denotes the maximum possible cumulative computational demand, normalized by interval length, of priority p or greater that is generated by real-time instance I. Clearly, $\mathsf{load}(p, I) \leq \mathsf{load}(I)$.

Finally, for each i, let Δ_i be defined as follows:

$$\Delta_i \stackrel{\text{def}}{=} \left(\max_{\rho(J_k) \geq \rho(J_i)} \{D_k\} \right) / D_i, \tag{6.2}$$

i.e., Δ_i denotes the ratio of the largest deadline parameter of a job with priority

greater than or equal to J_i's priority, to J_i's deadline parameter. This parameter will prove useful in determining schedulability conditions for real-time instances.

6.2 DM-schedulability Conditions

We now derive sufficient conditions for determining whether a given real-time instance I is DM schedulable upon a specified number of processors. In addition, we will quantify the "goodness" of these schedulability conditions via a resource-augmentation metric. The first result that we obtain is valid for *all* fixed-job-priority scheduling algorithms. (Note that fixed-task-priority scheduling algorithms are special subset of fixed-job-priority scheduling algorithms; therefore, even though we have categorized DM as fixed-task-priority, it is a fixed-job-priority algorithm, as well).

Theorem 6.1 *Let I denote a real-time instance such that for each $J_i \in I$, the following condition is satisfied:*

$$\mathsf{load}(\rho(J_i), I) \leq \frac{1}{2\Delta_i + 1} \left(m - (m-1)\frac{E_i}{D_i} \right) \ . \tag{6.3}$$

Instance I is fixed job-priority schedulable (under the full-migration scheduling paradigm) upon a platform comprised of m unit-capacity processors. ■

Proof: We will prove the contrapositive. Suppose that a given real-time instance $I = \{J_1, J_2, \ldots\}$ is *un*schedulable (according to an arbitrary fixed-job-priority scheduling algorithm) upon m unit-capacity processors, and let J_i denote the first job which misses its deadline. Since J_i does not receive E_i units of execution over the interval $[A_i, A_i + D_i)$, it must be the case that jobs of equal or greater priority execute on all m processors for strictly more than $(D_i - E_i)$ time units during this interval. That is, such jobs of equal or greater priority execute within this interval for $> m \times (D_i - E_i)$

time units.

Let J_s denote the job with earliest arrival time that has priority $\geq \rho(J_i)$, such that $A_i < A_s + D_s$ and J_s is executed by the fixed-job-priority algorithm during the interval $[A_i, A_i + D_i)$; and let J_f denote the job with latest (absolute) deadline that has priority $\geq \rho(J_i)$, such that $A_f < A_i + D_i$ and J_f is executed by the fixed-job-priority algorithm during the interval $[A_i, A_i + D_i)$. At least $m \times (D_i - E_i)$ time units of execution occurring over $[A_i, A_i + D_i)$ are generated by jobs of priority $\geq \rho(J_i)$ that arrive in, and have their deadlines within $[A_s, A_f + D_f)$.

Recall, from Equation 6.2, that Δ_i denotes the ratio of the largest deadline parameter of a job with priority greater than or equal to J_i's priority, to J_i's deadline parameter. Hence, D_s and D_f are both $\leq D_i \times \Delta_i$. Therefore, the intervals $[A_s, A_i)$ and $[A_i + D_i, A_f + D_f)$ both have a respective length of at most $D_i \times \Delta_i$. It is evident that the interval $[A_s, A_f + D_f)$ is of length at most $(2\Delta_i + 1) \times D_i$. Hence, for J_i to miss its deadline, it is necessary that the demand of jobs over $[A_s, A_f + D_f)$ (not including J_i's contribution) satisfy the following inequality:

$$
\begin{aligned}
&(A_f + D_f - A_s)\mathsf{load}(\rho(J_i), I) - E_i > m(D_i - E_i) \\
\Rightarrow &(2\Delta_i + 1) \cdot D_i \cdot \mathsf{load}(\rho(J_i), I) - E_i > m(D_i - E_i) \\
&\equiv \mathsf{load}(\rho(J_i), I) > \tfrac{m - (m-1)\ \frac{E_i}{D_i}}{2\Delta_i + 1}. \qquad (6.4)
\end{aligned}
$$

We have seen above that, in order for J_i to miss its deadline, it is necessary that Condition 6.4 be satisfied; Theorem 6.1 below follows by noting that Condition 6.4 is the negation of Equation 6.3. ■

Observe that in DM-scheduling, jobs are assigned priorities in inverse proportion of their relative deadline parameter; therefore, for a job J_i, it must be the case that all higher-priority jobs J_k have $D_k \leq D_i$. Thus, $\Delta_i \leq 1$. The following corollary immediately follows from combining this observation and Theorem 6.1.

Corollary 6.1 *Let I be a real-time instance such that for all $J_i \in I$ the following condition is satisfied:*

$$\mathsf{load}(\rho(J_i), I) \leq \frac{1}{3}\left(m - (m-1)\frac{E_i}{D_i}\right) \ . \tag{6.5}$$

Instance I is DM *schedulable upon a platform comprised of m unit-capacity processors.*

■

§ **Resource Augmentation..** The following corollary gives a simple test for the DM-schedulability of a real-time instance I.

Corollary 6.2 *Any real-time instance I satisfying the following conditions*

$$\forall J_k \in I: \ \mathsf{load}(\rho(J_k), I) \ \leq \ \frac{m^2}{4m-1} \tag{6.6}$$

$$\mathsf{max\text{-}job\text{-}density}(I) \ \leq \ \frac{m}{4m-1} \tag{6.7}$$

is successfully scheduled by the deadline-monotonic scheduling algorithm upon an m-processor unit-capacity multiprocessor platform.

Proof: In order that instance I be successfully scheduled by the deadline-monotonic scheduling algorithm on m unit-capacity processors it is, by Equation 6.5, sufficient that for all jobs$J_k \in I$,

$$\begin{aligned}
& \mathsf{load}(I) \leq \frac{1}{3} \times (m - (m-1)\mathsf{max\text{-}job\text{-}density}(I)) \\
\Leftarrow \quad & \text{(From Condition 6.7)} \\
& \mathsf{load}(I) \leq \frac{1}{3} \times \left(m - (m-1)\frac{m}{4m-1}\right) \\
\equiv \quad & \mathsf{load}(I) \leq \frac{m^2}{4m-1},
\end{aligned}$$

which is true, (by Condition 6.6 above). ■

Clearly, Conditions 6.6 and 6.7 are necessary for any I to be schedulable upon a platform comprised of m processors each of computing capacity $m/(4m-1)$; by Corollary 6.2 above, these conditions are also sufficient for I to be schedulable upon m *unit*-capacity processors. Hence to within a constant factor of less than four, we have obtained bounds on the multiplicative speed-up needed for the conditions based on load and max-job-density to suffice for schedulability. The following theorem formally states this; the proof of the theorem is identical to Theorem 4.2.

Theorem 6.2 *Any real-time instance I that is* feasible *(according to some hypothetically optimal feasibility test) upon m-processors of unit capacity is guaranteed to satisfy the conditions of Equation 6.6 and 6.7 of Corollary 6.2 on an m-processor platform where each processor has speed $4 - \frac{1}{m}$.*

6.3 EDF-schedulability conditions

In the previous section, we were able to derive conditions for fixed-job-priority scheduling algorithms, in general (Theorem 6.1). To obtain these conditions, we needed to consider, for any job J_i, all the jobs with priority $> \rho(J_i)$, even with deadlines after J_i. For the EDF-scheduling algorithm, we only need to consider jobs with deadlines prior to J_i's. This allows us to derive the following sufficient conditions for EDF schedulability.

Theorem 6.3 *Let I denote a real-time instance such that for each $J_i \in I$, the following condition is satisfied:*

$$\mathsf{load}(\rho(J_i), I) \leq \frac{1}{\Delta_i + 1}\left(m - (m-1)\frac{E_i}{D_i}\right) \ . \tag{6.8}$$

Instance I is EDF *schedulable upon a platform comprised of m unit-capacity proces-*

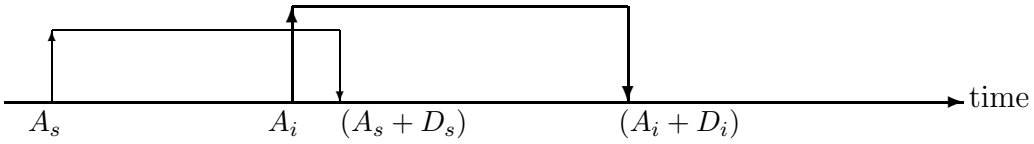


Figure 6.1: Job J_i is being considered. Job J_s is a job with priority $\geq \rho(J_i)$ with the earliest arrival time whose "intervals" overlap with that of J_i.

sors. ■

Proof:

We will prove the contrapositive of the theorem. Suppose that a given real-time instance $I = \{J_1, J_2, \ldots\}$ is *un*schedulable (according to the EDF scheduling algorithm) upon m unit-capacity processors, and let J_i denote the first job which misses its deadline. As in Theorem 6.1, since J_i does not receive E_i units of execution over the interval $[A_i, A_i + D_i)$, it must be the case that jobs of equal or greater priority execute on all m processors for strictly more than $(D_i - E_i)$ time units during this interval. Since in EDF-scheduling only jobs with earlier absolute deadlines have higher priority, at least $m(D_i - E_i)$ units of execution belongs to jobs that have their deadline in the interval $[A_i, A_i + D_i)$.

Let J_s denote the job with earliest arrival time that has priority $\geq \rho(J_i)$, such that $A_i < A_s + D_s \leq A_i + D_i$ (see Figure 6.1). It is evident from the figure that all the execution by jobs of priority $\geq \rho(J_i)$ occurring over $[A_i, A_i + D_i)$ must be generated by jobs that arrive in, and have their deadlines within, $[A_s, A_i + D_i)$.

D_s is $\leq D_i \times \Delta_i$. From Figure 6.1, it is clear that the interval $[A_s, A_i + D_i)$ is of length at most $(\Delta_i + 1) \times D_i$. Hence, for J_i to miss its deadline, it is necessary that th demand of jobs over $[A_s, A_i + D_i)$ (not including J_i's contribution) satisfy the

following inequality:

$$\begin{aligned}
&(A_i + D_i - A_s)\mathsf{load}(\rho(J_i), I) - E_i > m(D_i - E_i) \\
\Rightarrow &(\Delta_i + 1) \cdot D_i \cdot \mathsf{load}(\rho(J_i), I) - E_i > m(D_i - E_i) \\
&\equiv \mathsf{load}(\rho(J_i), I) > \frac{m-(m-1)\ \frac{E_i}{D_i}}{\Delta_i+1}. \qquad (6.9)
\end{aligned}$$

We have seen above that, in order for J_i to miss its deadline, it is necessary that Condition 6.9 be satisfied; Theorem 6.3 below follows by noting that Condition 6.9 is the negation of Equation 6.8. ■

If the ratio between the largest relative deadline and smallest relative deadline of any jobs of real-time instance I is bounded from above by constant K (i.e., for all $J_i \in I$, $\Delta_i \leq K$), then we may obtain a schedulability condition for EDF similar to Corollary 6.1:

Corollary 6.3 *Let I be a real-time instance such that for each $J_i \in I$ the following condition is satisfied:*

$$\mathsf{load}(\rho(J_i), I) \leq \frac{1}{K+1}\left(m - (m-1)\frac{E_i}{D_i}\right) \qquad (6.10)$$

where $K \stackrel{def}{=} \max_{J_i \in I}\{\Delta_i\}$. Then, instance I is EDF *schedulable upon a platform comprised of m unit-capacity processors.* ■

The above test works very well for small values of K. However, since the test of Corollary 6.3 is dependent on the ratio between the largest and smallest relative deadlines of I and this ratio may not be *a priori* bounded, a constant-factor resource-augmentation bound may not be derived from this test. Instead, we will now in the next subsection discuss a different EDF-schedulability test based on Corollary 4.2 that has a resource-augmentation approximation ratio.

6.3.1 A Different EDF-schedulability Test

As discussed in Chapter 2, Phillips et al. (Phillips et al., 1997) related the concept of multiprocessor feasibility with EDF schedulability using a technique known as *resource augmentation*. Specifically, they proved that any real-time instance I feasible on a platform with m unit-capacity processors is schedulable according to EDF on processing platform with m processors each of speed $(2 - \frac{1}{m})$. The following schedulability condition is immediately obtained from the feasibility test of Corollary 4.2 and using a resource augmentation speed-up result of (Phillips et al., 1997).

Corollary 6.4 *For real-time instance I and identical multiprocessor platform Π comprised of m unit-speed processors, if*

$$\mathsf{load}(I) \leq \frac{(\sqrt{2}-1)m^2}{2m-1} \quad \textit{and} \tag{6.11}$$

$$\mathsf{max\text{-}job\text{-}density}(I) \leq \frac{(\sqrt{2}-1)m}{2m-1}, \tag{6.12}$$

then the earliest-deadline-first scheduling algorithm (EDF) can successfully schedule I upon Π.

Proof: Consider a platform where each processor is of speed $\frac{m}{2m-1}$ times speed of platform Π (denote this platform as $\frac{m}{2m-1} \cdot \Pi$). Let I' denote real-time instance I normalized to the speed of $\frac{m}{2m-1} \cdot \Pi$; that is, for each job $J_i \in I$, there exists a job J'_i with the same arrival-time and deadline with $\frac{2m-1}{m} E_i$ execution requirement. Thus,

$$\mathsf{load}(I') = \frac{2m-1}{m} \cdot \mathsf{load}(I) \quad \text{and} \tag{6.13}$$

$$\mathsf{max\text{-}job\text{-}density}(I') = \frac{2m-1}{m} \cdot \mathsf{max\text{-}job\text{-}density}(I). \tag{6.14}$$

By Corollary 4.2, if $\mathsf{load}(I') \leq (\sqrt{2}-1)m$ and $\mathsf{max\text{-}job\text{-}density}(I') \leq (\sqrt{2}-1)$, then I' is feasible on $\frac{m}{2m-1} \cdot \Pi$. By Theorem 2.3, I' is also EDF-schedulable on Π. Substituting Equations 6.13 and 6.14 into these condition, we obtain Conditions 6.11 and 6.12 of the corollary.

■

Using the same techniques of Theorem 4.2, the following resource augmentation result immediately follows.

Theorem 6.4 *Any real-time instance I that is* feasible *(according to some hypothetically optimal feasibility test) upon m-processors of unit capacity is guaranteed to satisfy the conditions of Equation 6.6 and 6.7 of Corollary 6.2 on an m-processor platform where each processor has speed $2 + 2\sqrt{2} - \frac{\sqrt{2}+1}{m}$.*

6.4 Full-Migration Schedulability of Sporadic Task Systems

As shown in Section 3.4, the load and max-job-density of a sporadic task system can be efficiently determined. In this section, we give an application of the conditions for DM scheduling derived in Section 6.2 to the scheduling of sporadic task systems. We compare the schedulability tests of this dissertation to previously-known tests for the DM scheduling of sporadic task systems.

Consider a sporadic task system $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ comprised of n tasks. Without loss of generality, *assume that the tasks are indexed according to decreasing priorities*: task τ_1 is assigned the greatest priority, τ_n the least priority, and τ_k's priority is higher than τ_{k+1}'s for all k. We make no assumption about the relationship between a task's relative deadline and period parameter; however, if $d_k > p_k$, our current analysis assumes that it is possible for two jobs of τ_k to be active at the same time (i.e., for a

given time t, two or more jobs of τ_k may execute concurrently).

Let I denote any real-time instance that is generated during run-time by sporadic task system τ. Since the job J_i under consideration is assumed to have been generated by task τ_k, only tasks $\tau_1, \tau_2, \ldots, \tau_k$ will contribute to the $\mathsf{load}(\rho(J_i), I)$ term. The maximum cumulative execution requirement by jobs of these tasks over any time interval $[t_1, t_2)$ is at most the sum of the maximum execution requirements of the individual tasks:

$$\mathsf{demand}(\rho(J_i), I, t_1, t_2) \leq (\sum_{j=1}^{k} \mathrm{DBF}(\tau_j, t_2 - t_1)).$$

From the definition of load with respect to a given priority (Equation 6.1), it follows that

$$\mathsf{load}(\rho(J_i), I) \leq \max_{t \geq 0} \left\{ (\sum_{j=1}^{k} \mathrm{DBF}(\tau_j, t))/t \right\}. \tag{6.15}$$

Let us now overload the definition of load to apply to sporadic tasks of priority greater or equal to task τ_k as follows:

$$\mathsf{load}(k, \tau) \stackrel{\text{def}}{=} \max_{t \geq 0} \left\{ (\sum_{j=1}^{k} \mathrm{DBF}(\tau_j, t))/t \right\}.$$

That is, $\mathsf{load}(k, \tau)$ denotes the maximum cumulative computational demand, normalized by interval length, of priority k or greater that can be generated by sporadic task system τ.

Recall that only the jobs generated by tasks $\{\tau_1, \tau_2, \ldots, \tau_{k-1}\}$ have greater priority than a job of τ_k. Therefore, using the definition of load from Equation 6.15, we may derive the following additional corollary from Corollary 6.1.

Corollary 6.5 *Any sporadic task system* $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ *satisfying*

$$\forall k \ : 1 \leq k \leq n : \ \mathsf{load}(k, \tau) \leq \frac{1}{3}\left(m - (m-1)\frac{e_k}{d_k}\right) \tag{6.16}$$

is DM *schedulable upon a platform comprised of* m *unit-capacity processors.* ■

The same resource-augmentation technique, used in Section 6.2, can be applied to sporadic task systems to obtain a resource-augmentation bound of $4 - \frac{1}{m}$ for DM scheduling of sporadic tasks.

Recently, researchers have focussed on the multiprocessor static-priority scheduling of sporadic task systems. Baker (Baker, 2003; Baker, 2006a) and Bertogna et al. (Bertogna et al., 2005b) have derived sufficient conditions for the multiprocessor, static-priority schedulability of sporadic task systems.

For the sake of comparison, recall Theorem 2.7 from Chapter 2. Previous work has only empirically evaluated the effectiveness of the approach of Theorem 2.7, and, to the best of our knowledge, no resource-augmentation bounds have been obtained. In the next theorem, we give a lower bound on the resource-augmentation bound associated with Theorem 2.7:

Theorem 6.5 *The schedulability tests of Theorem 2.7 (Conditions 2.11 and 2.12) cannot have a resource-augmentation bound smaller than* $3 - \frac{2}{m}$ *for a platform with* m*-processors (where* $m > 1$*).*

Proof: Consider the following task system τ comprised of $(m-1)\ell$ small tasks and a single large task: the small tasks $\tau_i \in \{\tau_1, \ldots \tau_{(m-1)\ell}\}$ have specification $\tau_i \stackrel{\text{def}}{=} (e_i, d_i, p_i) = (1, \ell, \ell)$; the large task $\tau_{(m-1)\ell+1} \stackrel{\text{def}}{=} (\ell, \ell, \ell)$. This task system may be scheduled on a platform with m unit-capacity processors. It is easy to see that the $(m-1)\ell$ small tasks may be scheduled upon the first $(m-1)$ processors (ℓ fit on

each processor), and the large task is schedulable on the last processor. However, task $\tau_{(m-1)\ell+1}$ cannot be verified to be schedulable according to Theorem 4 for $\ell \geq 2$: observe that for all $i \in \{1, \ldots, (m-1)\ell\}$, $\beta_{(m-1)\ell+1}(i)$ equals $\frac{2}{\ell}$ (with respect to $\tau_{(m-1)\ell+1}$). Since $\beta_{(m-1)\ell+1}(i) = \frac{2}{\ell} > 1 - 1 = (1 - \frac{e_{(m-1)\ell+1}}{d_{(m-1)\ell+1}})$ for all $\ell \geq 2$, we must check Conditions 2.11 of Theorem 2.7 to verify if $\tau_{(m-1)\ell+1}$ is schedulable. This is equivalent to checking $\sum_{k=1}^{(m-1)\ell}(1-1) < m(1-1)$, which is obviously false.

We now determine the smallest constant $\alpha > 1$ such that $\tau_{(m-1)\ell+1}$ is schedulable on m α-speed processors according to Conditions 2.11 and 2.12 of Theorem 2.7. For $\tau_i \in \{\tau_1, \ldots \tau_{(m-1)\ell}\}$ and $\ell \geq 2$,

$$\begin{aligned} \beta_\ell(i) &= \frac{1 \cdot \frac{1}{\alpha} + \min\left(\frac{1}{\alpha}, \max\left(0, \ell - (1 \cdot \ell) + \ell - \frac{1}{\alpha}\right)\right)}{\ell} \\ &= \frac{2}{\alpha \cdot \ell}. \end{aligned}$$

If $\beta_{(m-1)\ell+1}(i) > 1 - \frac{e_{(m-1)\ell+1}}{\alpha \cdot d_{(m-1)\ell+1}}$, then to determine if $\tau_{(m-1)\ell+1}$ is schedulable according to Theorem 2.7, we must check Condition 2.11:

$$\begin{aligned} \textstyle\sum_{i=1}^{(m-1)\ell}(1 - \frac{1}{\alpha}) &< m(1 - \frac{1}{\alpha}) \\ \Rightarrow \qquad (m-1)\ell &< m. \end{aligned}$$

The last inequality is false for all $\ell \geq 2$ and $m > 1$; thus, if $\beta_{(m-1)\ell+1}(i) > 1 - \frac{e_{(m-1)\ell+1}}{\alpha \cdot d_{(m-1)\ell+1}}$, the schedulability of $\tau_{(m-1)\ell+1}$ cannot be verified by Theorem 2.7 for any speedup $\alpha > 1$. So, we will assume that $\beta_{(m-1)\ell+1}(i) \leq 1 - \frac{e_{(m-1)\ell+1}}{\alpha \cdot d_{(m-1)\ell+1}}$ for all $\tau_i \in \{\tau_1, \ldots \tau_{(m-1)\ell}\}$ and $\ell \geq 2$. Thus, to find the smallest α that satisfies Conditions 2.11 or 2.12 (with respect to the schedulability of $\tau_{(m-1)\ell+1}$), we must solve the following equation obtained from the conditions of Theorem 2.7:

$$m(1 - \frac{1}{\alpha}) \geq (m-1) \cdot \ell \cdot \frac{2}{\alpha \cdot \ell}. \tag{6.17}$$

Solving for $\alpha > 1$, we find that $\alpha \geq 3 - \frac{2}{m}$ which is a lower bound on the resource-

augmentation factor needed by Conditions 2.11 and 2.12. ■

The previous theorem only provides a lower bound on the resource-augmentation factor, while Corollary 6.2 gives an upper bound. It is interesting to note that the test of Corollary 6.5 applied to the example task system used in the previous proof requires a speed-up of $4 - \frac{1}{m}$. Further work is needed to obtain an upper bound on the resource-augmentation factor for the test of Theorem 2.7.

6.5 Summary

The results of this chapter have shown that despite the non-existence of optimal multiprocessor scheduling algorithms, online algorithms with constant-factor resource-augmentation approximation ratios exist. We have derived sufficient conditions for the well-known EDF and DM scheduling algorithms in terms of load and max-job-density. For the multiprocessor scheduling, these results are the first known schedulability conditions with resource-augmentation approximation ratios. Future work will explore whether these conditions may be tightened and if there exists scheduling algorithms with better resource augmentation bounds.

Chapter 7

The Partitioned Scheduling and Schedulability Analysis of Sporadic Task Systems

In this chapter, we report our findings concerning the preemptive multiprocessor scheduling of sporadic real-time systems under the partitioned paradigm. Observe that the process of partitioning tasks among processors reduces a multiprocessor scheduling problem to a series of uniprocessor problems (one to each processor). In this chapter, we consider both fixed-task-priority and fixed-job-priority scheduling algorithms at the uniprocessor scheduling level. For fixed-job-priority algorithms, the optimality of EDF for preemptive uniprocessor scheduling (Liu and Layland, 1973; Dertouzos, 1974) makes EDF a reasonable algorithm to use as the run-time scheduling algorithm on each processor. For fixed-task-priority algorithms, DM is known to be optimal for special subclasses of sporadic task systems on uniprocessors (Leung and Whitehead, 1982). Therefore, we consider both of these algorithms in this chapter.

Throughout this chapter, we will refer to special subclasses of sporadic task systems that are classified by the relationship between the values of p_i and d_i for each

$\tau_i \in \tau$. For the purposes of this chapter, we consider three subclasses based on this relationship.

- **Implicit-deadline**: Each sporadic task $\tau_i \in \tau$ satisfies the constraint that $d_i = p_i$.
- **Constrained**: Each sporadic task $\tau_i \in \tau$ satisfies the constraint that $d_i \leq p_i$.
- **Arbitrary**: There is no restriction placed on the relationship between d_i and p_i.

§ **Summary of Contributions.** For EDF-scheduled processors, we propose two polynomial-time partitioning algorithms. The first algorithm we propose assigns sporadic tasks to processors based on their *task density* parameter (density is defined as the execution requirement of a task divided by the minimum of either the period or relative deadline parameter). The second partitioning algorithm we propose uses an approximation to the demand-bound function (similar to the one defined in Section 3.4) for a sporadic task and task utilization (utilization is defined as execution requirement divided by the period parameter) as dual criteria for assigning a task to a processor. For DM-scheduled processors, we propose a polynomial-time partitioning algorithm which uses an approximation based on a different real-time workload characterization (not yet mentioned in this dissertation) called the *request-bound function*. All of the proposed partitioning algorithms are variants of the *First-Fit* bin-packing heuristic (Johnson, 1973).

For the partitioning algorithms using the demand-bound function and request-bound function approximation, we derive sufficient conditions for success of our algorithm (Theorems 7.4, 7.5, 7.8, and 7.9). We also show (Corollary 7.2 and Theorem 7.10) that our demand-based and request-bound function partitioning algorithms both have the following performance guarantee for arbitrary sporadic task systems.

> If a sporadic task system is feasible on m identical processors, then the same task system can be partitioned by our algorithm among m identical processors *in which the individual processors are* $(4-\frac{2}{m})$ *times as fast* as in the original system, such that all jobs of all tasks assigned to each processor will always meet their deadlines if scheduled using the preemptive EDF scheduling algorithm.

A slightly tightened guarantee may also be defined for the performance of the algorithms over constrained sporadic task systems.

For the density-based partitioning algorithm, we also derive sufficient conditions for success (Theorem 7.1). However, we show (Theorem 7.2) that density-based partitioning does not have a performance guarantee on the necessary speed of each processor for the partitioning algorithm to succeed (i.e., there cannot exist a guarantee for density-based partitioning similar to the preceding guarantee for demand-based partitioning).

§ **Organization.** The remainder of this chapter is organized as follows. In Section 7.1, we design partitioning algorithms for when EDF is used to schedule each individual processor. In Section 7.1.2 we present, prove the correctness of, and evaluate the performance of a very simple density-based partitioning algorithm. In Section 7.1.3, we present, and prove correct, a somewhat more sophisticated demand-based partitioning algorithm. In Section 7.1.4, we prove that this algorithm satisfies a property that the simpler algorithm does not possess: if given sufficiently faster processors, it is able to guarantee to meet all deadlines for all feasible systems. We also list some other results that we have obtained, and include a brief discussion of the significance of our results. In Section 7.1.5 we propose an improvement to the algorithm: although this improvement does not seem to effect the worst-case behavior of the algorithm, it is shown to be of use in successfully partitioning some sporadic

task systems that our basic algorithm — the one presented in Section 7.1.3 — fails to handle.

In Section 7.2, we design a partitioning algorithm for using DM-scheduling algorithm on each processor. In Section 7.2.2, we present our polynomial-time partitioning algorithm and prove its correctness. Section 7.2.3 evaluates the efficacy of the partitioning algorithm in terms of sufficient conditions for success and resource augmentation approximation bounds.

7.1 EDF-based Partitioning

Most partitioning schemes use the following steps at a high-level:

1. Sort tasks in order of some criteria.

2. In the sorted order of Step 1, assign each task to a processor upon which it "fits." A task "fits" on a processor if it will always meet all deadlines when assigned to the processor, and it does not cause another previously-assigned task to miss a deadline.

3. After each task has been assigned to a processor, use a uniprocessor scheduling algorithm on each processor to schedule the processor's respective tasks.

Observe that Steps 1 and 2 occur prior to system runtime. Step 3 occurs during runtime after task assignment.

A further remark about Step 2: a uniprocessor schedulability test is typically the process by which it is determined whether a task fits on a processor. In this section, we consider EDF scheduling. Two possible different EDF-schedulability tests are the demand-bound function or task density. In the next subsection (Section 7.1.1), we define an approximation to the demand-bound function and compare it with task

density. In Section 7.1.2, we define a polynomial-time partitioning algorithm using the density-based schedulability test. In Section 7.1.3, we define a polynomial-time partitioning algorithm using the DBF-approximation.

7.1.1 Approximation of $\text{DBF}(\tau_i, t)$

The function $\text{DBF}(\tau_i, t)$, if plotted as a function of t for a given task τ_i, is represented by a series of "steps," each of height e_i, at time-instants d_i, $d_i + p_i$, $d_i + 2p_i$, $\cdots$, $d_i + kp_i$, $\cdots$. As reviewed in Section 3.4, Albers and Slomka (Albers and Slomka, 2004) Slomka (Albers and Slomka, 2004) have proposed a technique for *approximating* the DBF, which tracks the DBF exactly through the first several steps and then approximates it by a line of slope e_i/p_i (see Figure 7.1). In the following, we are applying this technique in essentially tracking DBF exactly for a *single* step of height e_i at time-instant d_i, followed by a line of slope e_i/p_i; this is essentially $\text{DBF}(\tau_i, t, k_i)$ defined in Equation 3.8 with $k_i = 1$.

$$\text{DBF}^*(\tau_i, t) = \begin{cases} 0, & \text{if } t < d_i \\ e_i + u_i \times (t - d_i), & \text{otherwise} \end{cases} \tag{7.1}$$

As stated earlier, it has been shown that the cumulative execution requirement of jobs of τ_i over an interval is maximized if one job arrives at the start of the interval, and subsequent jobs arrive as rapidly as permitted. Intuitively, approximation DBF^* (Equation 7.1) models this job-arrival sequence by requiring that the first job's deadline be met explicitly by being assigned e_i units of execution between its arrival-time and its deadline, and that τ_i be assigned $u_i \times \delta\, t$ of execution over time-interval $[t, t + \delta\, t)$, for all instants t after the deadline of the first job, and for arbitrarily small positive $\delta\, t$ (see Figure 7.2 for a pictorial depiction).

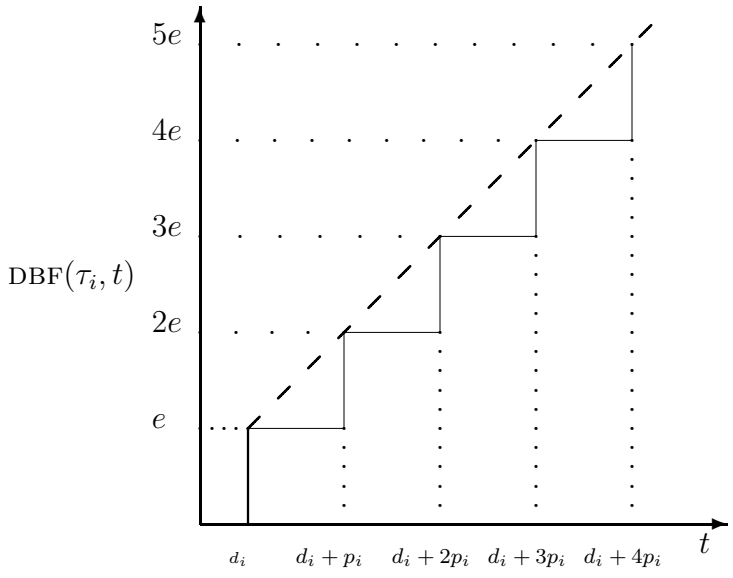


Figure 7.1: The step function denotes a plot of $\text{DBF}(\tau_i, t)$ as a function of t. The dashed line represents the function $\text{DBF}^*(\tau_i, t)$, approximating $\text{DBF}(\tau_i, t)$; for $t < d_i$, $\text{DBF}^*(\tau_i, t) \equiv \text{DBF}(\tau_i, t) = 0$.

Observe that the following inequalities hold for all τ_i and for all $t \geq 0$:

$$\text{DBF}(\tau_i, t) \leq \text{DBF}^*(\tau_i, t) < 2 \cdot \text{DBF}(\tau_i, t) \tag{7.2}$$

with the ratio $\text{DBF}^*(\tau_i, t)/\text{DBF}(\tau_i, t)$ being maximized just prior to the deadline of the second job of τ_i — i.e., at $t = d_i + p_i - \epsilon$ for ϵ an arbitrarily small positive number — in the synchronous arrival sequence; at this time-instant, $\text{DBF}^*(\tau_i, t) \to 2e$ while $\text{DBF}(\tau_i, t) = e$.

§ **Comparison of DBF^* and the density approximation.** It is known that a sufficient condition for a sporadic task system to be feasible upon a unit-capacity *uni*processor is $(\sum_{\tau_i \in \tau} \textsf{task-density}(\tau_i)) \leq 1$; i.e., the sum of the densities of all tasks in the system not exceed one (see, e.g., (Liu, 2000, Theorem 6.2)). This condition is obtained by essentially approximating the demand bound function $\text{DBF}(\tau_i, t)$ of τ_i by the quantity $(t \cdot \textsf{task-density}(\tau_i))$. This approximation is never superior to our

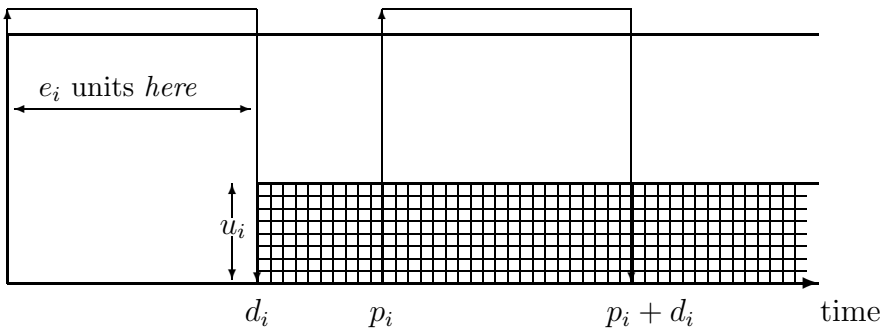


Figure 7.2: Pictorial representation of task τ_i's reservation of computing capacity in a processor-sharing schedule. e_i units of execution are reserved over the interval $[0, d_i)$. The hatched region depicts τ_i's reservation of computing capacity over the interval $[d_i, \infty)$ – over any time interval $[t, t + \delta\, t)$, an amount $\delta\, t \times u_i$ of computing capacity is reserved.

approximation DBF*, and is inferior if $d_i \neq p_i$: for our example in Figure 7.1, this approximation would be represented by a straight line with slope e_i/d_i passing through the origin.

7.1.2 Density-Based Partitioning

In this section, we present a simple, efficient, algorithm for partitioning a sporadic task system among the processors of a multiprocessor platform. We also show that this partitioning algorithm does not offer a resource-augmentation performance guarantee.

Let us suppose that we are given sporadic task system τ comprised of n tasks $\tau_1, \tau_2, \ldots \tau_n$, and a platform comprised of m unit-capacity processors $\pi_1, \pi_2, \ldots, \pi_m$. For each task τ_i, recall that its **density** $\mathsf{task\text{-}density}(\tau_i)$ is defined as follows:

$$\mathsf{task\text{-}density}(\tau_i) \stackrel{\text{def}}{=} \frac{e_i}{\min(d_i, p_i)},$$

and

$$\mathsf{max\text{-}task\text{-}density}(\tau) \stackrel{\text{def}}{=} \max_{\tau_i \in \tau} \left(\mathsf{task\text{-}density}(\tau_i) \right).$$

Without loss of generality, let us assume that the tasks in τ are indexed according to non-increasing density: $\mathsf{task\text{-}density}(\tau_i) \geq \mathsf{task\text{-}density}(\tau_{i+1})$ for all i, $1 \leq i < n$. Our partitioning algorithm considers the tasks in the order $\tau_1, \tau_2, \ldots$. Suppose that tasks τ_1, τ_2, ..., τ_{i-1} have all been successfully allocated among the m processors, and we are now attempting to allocate task τ_i to a processor. Our algorithm for doing this is a variant of the *First Fit* (Johnson, 1974) algorithm for bin-packing, and is as follows (see Figure 7.3 for a pseudo-code representation). For any processor π_ℓ, let $\tau(\pi_\ell)$ denote the tasks from among $\tau_1, \ldots, \tau_{i-1}$ that have already been allocated to processor π_ℓ. Considering the processors $\pi_1, \pi_2, \ldots, \pi_m$, in any order, we will assign task τ_i to any processor π_k, $1 \leq k \leq m$, that satisfies the following condition:

$$\mathsf{task\text{-}density}(\tau_i) + \left(\sum_{\tau_j \in \tau(\pi_k)} \mathsf{task\text{-}density}(\tau_j) \right) \leq 1. \tag{7.3}$$

Lemma 7.1 *Algorithm* densityPartition *successfully partitions any sporadic task system τ on $m \geq 1$ processors, that satisfies the following condition:*

$$\sum_{\tau_i \in \tau} \mathsf{task\text{-}density}(\tau_i) \leq m - (m-1) \times \mathsf{max\text{-}task\text{-}density}(\tau). \tag{7.4}$$

Proof: Let us suppose that Algorithm densityPartition fails to partition task system τ; in particular, let us assume that it fails to assign task τ_i to any processor, for some $i \leq n$. It must be the case that each of the m processors fails the test of Equation 7.3; i.e., tasks previously assigned to each processor have their densities summing to more than $(1 - \mathsf{task\text{-}density}(\tau_i))$. Summing over all m processors, this

```
densityPartition(τ, m)
    ▷ The collection of sporadic tasks τ = {τ_1, ..., τ_n} is to be partitioned on
      m identical, unit-capacity processors denoted π_1, π_2, ..., π_m. (Tasks
      are indexed according to non-increasing density: task-density(τ_i) ≥
      task-density(τ_{i+1}) for all i.)  τ(π_k) denotes the tasks assigned to pro-
      cessor π_k; initially, τ(π_k) ← ∅ for all k.
1   for i ← 1 to n
    ▷ i ranges over the tasks
2           for k ← 1 to m
            ▷ k ranges over the processors, considered in any order
3                   if τ_i satisfies Conditions 7.3 on processor π_k then
                            ▷ assign τ_i to π_k; proceed to next task
4                           τ(π_k) ← τ(π_k) ⋃ {τ_i}
5                           break;
6           end (of inner for loop);
7           if (k > m) return PARTITIONING FAILED
8   end (of outer for loop);
9   return PARTITIONING SUCCEEDED
```

Figure 7.3: Pseudo-code for density-based partitioning algorithm.

implies that

$$\begin{aligned}
&\sum_{j=1}^{i-1} \textsf{task-density}(\tau_j) > m \times (1 - \textsf{task-density}(\tau_i)) \\
&\quad \equiv \quad \sum_{j=1}^{i} \textsf{task-density}(\tau_j) > m - (m-1)\textsf{task-density}(\tau_i) \\
&\quad \Rightarrow \quad \sum_{\tau_i \in \tau} \textsf{task-density}(\tau_i) > m - (m-1) \times \textsf{max-task-density}(\tau).
\end{aligned}$$

Hence, any system which Algorithm densityPartition fails to partition must have $\sum_{\tau_i \in \tau} \textsf{task-density}(\tau_i) > m - (m-1) \times \max_{\tau_i \in \tau} (\textsf{task-density}(\tau_i))$. The lemma follows.

■

Theorem 7.1 *Algorithm* densityPartition *successfully partitions any sporadic task*

system τ on $m \geq 2$ processors, that satisfies the following condition:

$$\sum_{i=1}^{n} \textsf{task-density}(\tau_i) \leq \begin{cases} m - (m-1)\textsf{max-task-density}(\tau) & \textit{if } \textsf{max-task-density}(\tau) \leq \frac{1}{2} \\ \frac{m}{2} + \textsf{max-task-density}(\tau) & \textit{if } \textsf{max-task-density}(\tau) \geq \frac{1}{2}. \end{cases} \tag{7.5}$$

Proof: Let us define a task τ_i to be **dense** if $\textsf{task-density}(\tau_i) > 1/2$. We consider two cases separately.

§ **Case 1: There are no dense tasks.** In this case $\textsf{max-task-density}(\tau) \leq 1/2$, and Equation 7.5 reduces to

$$\sum_{i=1}^{n} \textsf{task-density}(\tau_i) \leq m - (m-1) \times \textsf{max-task-density}(\tau);$$

by Lemma 7.1, this condition is sufficient to guarantee that Algorithm densityPartition successfully partitions the tasks in τ.

§ **Case 2: There are dense tasks.** In this case $\textsf{max-task-density}(\tau) > 1/2$, and Equation 7.5 reduces to

$$\sum_{i=1}^{n} \textsf{task-density}(\tau_i) \leq \frac{m}{2} + \textsf{max-task-density}(\tau). \tag{7.6}$$

Observe first that any task system satisfying Condition 7.6 has at most m dense tasks – this follows from the observation that one task will have density equal to $\textsf{max-task-density}(\tau)$, and at most $(m-1)$ tasks may each have density greater than $\frac{1}{2}$ while observing the $\sum_{i=1}^{n} \textsf{task-density}(\tau_i)$ bound of Condition 7.6. We consider separately the two cases when there are strictly less than m dense tasks, and when there are exactly m dense tasks.

§ **Case 2.1: Fewer than m dense tasks.** Let n_h denote the number of dense tasks. Algorithm densityPartition assigns each of these n_h tasks to a different processor.

We will apply Lemma 7.1 to prove that Algorithm densityPartition successfully assigns the tasks $\{\tau_{n_h+1}, \tau_{n_h+2}, \ldots, \tau_n\}$ on $(m - n_h)$ processors; the correctness of the theorem would then follow from the observation that Algorithm densityPartition does in fact have $(m - n_h)$ processors left over after tasks τ_1 through τ_{n_h} have been assigned.

Let us now compute upper bounds on the cumulative and maximum densities of the task system $\{\tau_{n_h+1}, \tau_{n_h+2}, \ldots, \tau_n\}$:

$$\begin{aligned}
&\sum_{j=n_h+1}^{n} \textsf{task-density}(\tau_j) = \sum_{j=1}^{n} \textsf{task-density}(\tau_j) - \sum_{j=1}^{n_h} \textsf{task-density}(\tau_j) \\
\Rightarrow \quad &\sum_{j=n_h+1}^{n} \textsf{task-density}(\tau_j) \leq \frac{m}{2} + \textsf{max-task-density}(\tau) - \sum_{j=1}^{n_h} \textsf{task-density}(\tau_j) \\
\Rightarrow \quad &\text{(from the fact that } \textsf{task-density}(\tau_1) = \textsf{max-task-density}(\tau) \\
&\qquad\qquad \text{and } \lambda \geq \frac{1}{2} \text{ for all } i = 2, \ldots, n_h) \\
&\sum_{j=n_h+1}^{n} \textsf{task-density}(\tau_j) \leq \frac{m}{2} + \textsf{max-task-density}(\tau) \\
&\qquad\qquad -(\textsf{max-task-density}(\tau) + \frac{1}{2} \times (n_h - 1)) \\
\equiv \quad &\sum_{j=n_h+1}^{n} \textsf{task-density}(\tau_j) \leq \frac{m - n_h + 1}{2}. \qquad (7.7)
\end{aligned}$$

Furthermore, since each task in $\{\tau_{n_h+1}, \tau_{n_h+2}, \ldots, \tau_n\}$ is, by definition of n_h, not dense, it is the case that

$$\max_{j=n_h+1}^{n} \Big(\textsf{task-density}(\tau_j)\Big) \leq \frac{1}{2}. \qquad (7.8)$$

By applying the bounds derived in Equation 7.7 and Equation 7.8 above to Lemma 7.1, we may conclude that Algorithm densityPartition successfully assigns the tasks

$\{\tau_{n_h+1}, \tau_{n_h+2}, \ldots, \tau_n\}$ on $(m - n_h)$ processors:

$$\begin{aligned}
&\sum_{j=n_h+1}^{n} \textsf{task-density}(\tau_j) \leq (m - n_h) - \\
&\qquad (m - n_h - 1) \max_{j=n_h+1}^{n} \Big(\textsf{task-density}(\tau_j)\Big) \\
\Leftarrow \quad &\frac{m - n_h + 1}{2} \leq (m - n_h) - \frac{m - n_h - 1}{2} \\
\equiv \quad &\frac{m - n_h + 1}{2} \leq \frac{m - n_h + 1}{2}
\end{aligned}$$

which is, of course, always true.

§ **Case 2.2: Exactly m dense tasks.** Let $\textsf{task-density}(\tau_R)$ denote the cumulative densities of all the non-dense tasks: $\textsf{task-density}(\tau_R) \stackrel{\text{def}}{=} \sum_{j=m+1}^{n} \textsf{task-density}(\tau_j)$. Observe that $\textsf{task-density}(\tau_R) < \frac{m}{2} - (m-1) \times \frac{1}{2} = \frac{1}{2}$. By Equation 7.5, since $\textsf{max-task-density}(\tau) > \frac{1}{2}$, the total system density does not exceed $\frac{m}{2} + \textsf{max-task-density}(\tau)$. Because $\textsf{task-density}(\tau_1) = \textsf{max-task-density}(\tau)$, the $(m-1)$ tasks $\tau_2, \tau_3, \ldots, \tau_m$ have total density $\leq \frac{m}{2} - \textsf{task-density}(\tau_R)$, it must be the case that $\textsf{task-density}(\tau_m)$ is at most $(\frac{m}{2} - \textsf{task-density}(\tau_R))/(m-1)$. We will prove that $\textsf{task-density}(\tau_m) + \textsf{task-density}(\tau_R) \leq 1$, from which it will follow that all the tasks $\tau_m, \ldots, \tau_n$ fit on a single processor. Indeed,

$$\begin{aligned}
&\textsf{task-density}(\tau_m) + \textsf{task-density}(\tau_R) \leq 1 \\
\Leftarrow \quad &\frac{\frac{m}{2} - \textsf{task-density}(\tau_R)}{m-1} + \textsf{task-density}(\tau_R) \leq 1 \\
\equiv \quad &m - 2\textsf{task-density}(\tau_R) + 2m\textsf{task-density}(\tau_R) - 2\textsf{task-density}(\tau_R) \leq 2m - 2 \\
\equiv \quad &\textsf{task-density}(\tau_R)(2m-4) \leq m - 2 \\
\equiv \quad &\textsf{task-density}(\tau_R) \leq \frac{1}{2},
\end{aligned}$$

which is true.

■

7.1.2.1 Resource Augmentation Analysis

Consider the task system τ comprised of the n tasks $\tau_1, \tau_2, \ldots, \tau_n$, with τ_i having the following parameters:

$$e_i = 2^{i-1}, \quad d_i = 2^i - 1, \quad p_i = \infty.$$

Observe that τ is feasible on a single unit-capacity processor, since

$$\sum_{i=1}^{n} \text{DBF}(\tau_i, t) = \sum_{i=1}^{\lfloor \log_2(t+1) \rfloor} 2^{i-1},$$

which is $\leq t$ for all $t \geq 0$, with equality at $t = 2 - 1, 2^2 - 1, \ldots, 2^n - 1$ and strict inequality for all other t.

Now, $\textsf{task-density}(\tau_i) > \frac{1}{2}$ for each i; i.e., each task is dense. Hence, Algorithm densityPartition will assign each task to a distinct processor, and hence is only able to schedule τ upon n unit-capacity processors. Alternatively, Algorithm densityPartition would need a processor of computing capacity $\geq n$ in order to have Condition 7.3 be satisfied for all tasks on a single processor. By making n arbitrarily large, we have the following result:

Theorem 7.2 *For any constant $\xi \geq 1$, there is a sporadic task system τ and some positive integer m such that τ is (global or partitioned) feasible on m unit-capacity processors, but Algorithm* densityPartition *fails to successfully partition the tasks in τ among* **(i)** *$\xi \times m$ unit-capacity processors, or* **(ii)** *m processors each of computing capacity ξ.* ■

7.1.3 Algorithm EDF-PARTITION

Given sporadic task system τ comprised of n tasks $\tau_1, \tau_2, \ldots \tau_n$, and a platform comprised of m unit-capacity processors $\pi_1, \pi_2, \ldots, \pi_m$, we now describe another algorithm for partitioning the tasks in τ among the m processors. In Section 7.1.4 we will prove that this algorithm, unlike the one described above in Section 7.1.2, does make non-trivial resource-augmentation performance guarantees. With no loss of generality, let us assume that the tasks in τ are indexed according to non-decreasing order of their relative deadline parameter (i.e., $d_i \leq d_{i+1}$ for all i, $1 \leq i < n$). Our partitioning algorithm (see Figure 7.4 for a pseudo-code representation) considers the tasks in the order $\tau_1, \tau_2, \ldots$. Suppose that tasks τ_1, τ_2, ..., τ_{i-1} have all been successfully allocated among the m processors, and we are now attempting to allocate task τ_i to a processor. For any processor π_ℓ, let $\tau(\pi_\ell)$ denote the tasks from among $\tau_1, \ldots, \tau_{i-1}$ that have already been allocated to processor π_ℓ. Considering the processors $\pi_1, \pi_2, \ldots, \pi_m$, in any order, we will assign task τ_i to the first processor π_k, $1 \leq k \leq m$, that satisfies the following two conditions:

$$\left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \right) \geq e_i \tag{7.9}$$

and

$$\left(1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \right) \geq u_i. \tag{7.10}$$

If no such π_k exists, then we declare failure: we are unable to conclude that sporadic task system τ is feasible upon the m-processor platform.

The following lemma asserts that, in assigning a task τ_i to a processor π_k, our partitioning algorithm does not adversely affect the feasibility of the tasks assigned earlier to each processor.

```
EDF-PARTITION(τ, m)
    ▷ The collection of sporadic tasks τ = {τ_1, ..., τ_n} is to be partitioned on
      m identical, unit-capacity processors denoted π_1, π_2, ..., π_m. (Tasks are
      indexed according to non-decreasing value of relative deadline parame-
      ter: d_i ≤ d_{i+1} for all i.) τ(π_k) denotes the tasks assigned to processor
      π_k; initially, τ(π_k) ← ∅ for all k.
1   for i ← 1 to n
    ▷ i ranges over the tasks, which are indexed by non-decreasing value of
      the deadline parameter
2           for k ← 1 to m
            ▷ k ranges over the processors, considered in any order
3                   if τ_i satisfies Conditions 7.9 and 7.10 on processor π_k then
                            ▷ assign τ_i to π_k; proceed to next task
4                           τ(π_k) ← τ(π_k) ⋃ {τ_i}
5                           break;
6           end (of inner for loop)
7           if (k > m) return PARTITIONING FAILED
8   end (of outer for loop)
9   return PARTITIONING SUCCEEDED
```

Figure 7.4: Pseudo-code for partitioning algorithm.

Lemma 7.2 *If the tasks previously assigned to each processor were* EDF*-feasible on that processor and our algorithm assigns task τ_i to processor π_k, then the tasks assigned to each processor (including processor π_k) remain* EDF*-feasible on that processor.*

Proof: Observe that the EDF-feasibility of the processors other than processor π_k is not affected by the assignment of task τ_i to processor π_k. It remains to demonstrate that, if the tasks assigned to π_k were EDF-feasible on π_k prior to the assignment of τ_i and Conditions 7.9 and 7.10 are satisfied, then the tasks on π_k remain EDF-feasible after adding τ_i.

The scheduling of processor π_k after the assignment of task τ_i to it is a uniprocessor scheduling problem. It is known (see, e.g. (Baruah et al., 1990b)) that a uniprocessor system of preemptive sporadic tasks is feasible if and only all deadlines can be met for the synchronous arrival sequence (i.e., when each task has a job arrive at the same

time-instant, and subsequent jobs arrive as rapidly as legal). Also, recall that EDF is an optimal preemptive uniprocessor scheduling algorithm. Hence to demonstrate that π_k remains EDF-feasible after adding task τ_i to it, it suffices to demonstrate that all deadlines can be met for the synchronous arrival sequence. Our proof of this fact is by contradiction. That is, we suppose that a deadline is missed at some time-instant t_f, when the synchronous arrival sequence is scheduled by EDF, and derive a contradiction which leads us to conclude that this supposition is incorrect, i.e., no deadline is missed.

Observe that t_f must be $\geq d_i$, since it is assumed that the tasks assigned to π_k are EDF-feasible prior to the addition of τ_i, and τ_i's first deadline in the critical arrival sequence is at time-instant d_i.

By the *processor demand criterion* for preemptive uniprocessor feasibility (see, e.g.,
(Baruah et al., 1990b)), it must be the case that

$$\text{DBF}(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}(\tau_j, t_f) > t_f \ ,$$

from which it follows, since DBF^* is always an upper bound on DBF, that

$$\text{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t_f) > t_f \ . \tag{7.11}$$

Since tasks are considered in order of non-decreasing relative deadline, it must be the case that all tasks $\tau_j \in \tau(\pi_k)$ have $d_j \leq d_i$. We therefore have, for each $\tau_j \in \tau(\pi_k)$,

$$\begin{aligned} \text{DBF}^*(\tau_j, t_f) &= e_j + u_j(t_f - d_j) \qquad \text{(By definition)} \\ &= e_j + u_j(d_i - d_j) + u_j(t_f - d_i) \\ &= \text{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i). \end{aligned} \tag{7.12}$$

Furthermore,

$$\begin{aligned}
&\text{DBF}^*(\tau_i, t_f) + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, t_f) \\
&\equiv \quad (e_i + u_i(t_f - d_i)) + \qquad \textit{(By Equation 7.12 above)} \\
&\qquad \left(\sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) + u_j(t_f - d_i) \right) \\
&\equiv \quad \left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \right) \\
&\qquad +(t_f - d_i) \left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j \right).
\end{aligned}$$

Consequently, Inequality 7.11 above can be rewritten as follows:

$$\begin{aligned}
&\left(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \right) + \\
&\qquad (t_f - d_i) \left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j \right) > (t_f - d_i) + d_i. \qquad (7.13)
\end{aligned}$$

However by Condition 7.9, $(e_i + \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i)) \leq d_i)$; Inequality 7.13 therefore implies

$$(t_f - d_i) \left(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j \right) > (t_f - d_i),$$

which in turn implies that

$$(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j) > 1,$$

which contradicts Condition 7.10. ■

In the special case where the given sporadic task system is known to be *constrained* — i.e., each task's relative deadline parameter is no larger than its period parameter — Lemma 7.3 below asserts that it actually suffices to test only Condition 7.9, rather

than having to test both Condition 7.9 and Condition 7.10.

Lemma 7.3 *For constrained sporadic task systems, any τ_i satisfying Condition 7.9 during the execution of Line 3 in the algorithm of Figure 7.4 satisfies Condition 7.10 as well.*

Proof: To see this, observe (from Equation 7.1) that for any constrained task τ_k, for all $t \geq d_k$ it is the case that

$$\text{DBF}^*(\tau_k, t) = u_k \times (t + p_k - d_k) \geq u_k \times t.$$

Hence,

$$\begin{aligned}
&\text{Condition 7.9} \\
\equiv \quad & \left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{DBF}^*(\tau_j, d_i) \geq e_i \right) \\
\Rightarrow \quad & d_i - \sum_{\tau_j \in \tau(\pi_k)} (u_j \times d_i) \geq e_i \\
\Rightarrow \quad & 1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq \frac{e_i}{d_i} \\
\Rightarrow \quad & 1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq u_i \\
&\equiv \text{Condition 7.10.}
\end{aligned}$$

■

Hence, if the task system τ being partitioned is known to be a constrained task system, we need only check Condition 7.9 (rather than both Condition 7.9 and Condition 7.10) on line 3 in Figure 7.4. The correctness of the partitioning algorithm follows, by repeated applications of Lemma 7.2.

Theorem 7.3 *If our partitioning algorithm returns* PARTITIONING SUCCEEDED *on task system τ, then the resulting partitioning is* EDF*-feasible.*

Proof Sketch: Observe that the algorithm returns PARTITIONING SUCCEEDED if and only if it has successfully assigned each task in τ to some processor.

Prior to the assignment of task τ_1, each processor is trivially EDF-feasible. It follows from Lemma 7.2 that all processors remain EDF-feasible after each task assignment assignment as well. Hence, all processors are EDF-feasible once all tasks in τ have been assigned. ■

Time Complexity

In attempting to map task τ_i, observe that our partitioning algorithm essentially evaluates, in Equations 7.9 and 7.10, the workload generated by the previously-mapped $(i-1)$ tasks on each of the m processors. Since $\text{DBF}^*(\tau_j, t)$ can be evaluated in constant time (see Equation 7.1), a straightforward computation of this workload would require $\mathcal{O}(i+m)$ time. Hence, the run-time of the algorithm in mapping all n tasks is no more than $\sum_{i=1}^{n} \mathcal{O}(i+m)$, which is $\mathcal{O}(n^2)$ under the reasonable assumption that $m \leq n$.

7.1.4 Evaluation

Our DBF^*-based partitioning algorithm represents a sufficient, rather than exact, test for feasibility — it is possible that there are systems that are feasible under the partitioned paradigm but which will be incorrectly flagged as "infeasible" by our partitioning algorithm. Indeed, this is to be expected since a simpler problem – partitioning collections of sporadic tasks that all have their deadline parameters equal to their period parameters – is known to be NP-hard in the strong sense while our algorithm runs in $\mathcal{O}(n^2)$ time. In this section, we offer a quantitative evaluation of the efficacy of our algorithm. Specifically, we derive some properties (Theorem 7.5 and Corollary 7.2) of our partitioning algorithm, which characterize its performance. We would

like to stress that *these properties are not intended to be used as feasibility tests to determine whether our algorithm would successfully schedule a given sporadic task system* – since our algorithm itself runs efficiently in polynomial time, the "best" (i.e., most accurate) polynomial-time test for determining whether a particular system is successfully scheduled by our algorithm is to actually run the algorithm and check whether it performs a successful partition or not. Rather, these properties are intended to provide a quantitative measure of how effective our partitioning algorithm is *vis a vis* the performance of an optimal scheduler.

The parameters that are useful to characterize the behavior of our algorithm are: $\mathsf{load}(\tau)$, $\mathsf{max\text{-}job\text{-}density}(\tau)$, $\mathsf{system\text{-}util}(\tau)$, and $\mathsf{max\text{-}util}(\tau)$ (these are defined in Chapters 2 and 3). Intuitively, the larger of $\mathsf{max\text{-}job\text{-}density}(\tau)$ and $\mathsf{max\text{-}util}(\tau)$ represents the maximum computational demand of any *individual* task, and the larger of $\mathsf{load}(\tau)$ and $\mathsf{system\text{-}util}(\tau)$ represents the maximum cumulative computational demand of all the tasks in the system.

Lemma 7.4 follows immediately.

Lemma 7.4 *If task system τ is feasible (under either the partitioned or the global paradigm) on an identical multiprocessor platform comprised of m_o processors of computing capacity ξ each, it must be the case that*

$$\xi \geq \max(\mathsf{max\text{-}job\text{-}density}(\tau), \mathsf{max\text{-}util}(\tau)),$$

and

$$m_o \cdot \xi \geq \max(\mathsf{load}(\tau), \mathsf{system\text{-}util}(\tau)).$$

Proof: Observe that

1. Each job of each task of τ can receive at most $\xi \cdot d_i$ units of execution by its deadline; hence, we must have $e_i \leq \xi \cdot d_i$.

2. No individual task's utilization may exceed the computing capacity of a processor; i.e., it must be the case that $u_i \leq \xi$.

Taken over all tasks in τ, these observations together yield the first condition.

In the second condition, the requirement that $m_o\xi \geq \mathsf{system\text{-}util}(\tau)$ simply reflects the requirement that the cumulative utilization of all the tasks in τ not exceed the computing capacity of the platform. The requirement that $m_o\xi \geq \mathsf{load}(\tau)$ is obtained by considering a sequence of job arrivals for τ that defines $\mathsf{load}(\tau)$; i.e., a sequence of job arrivals over an interval $[0, t_o)$ such that $\frac{\sum_{j=1}^{n} \mathrm{DBF}(\tau_j, t_o)}{t_o} = \mathsf{load}(\tau)$. The total amount of execution that all these jobs may receive over $[0, t_o)$ is equal to $m_o \cdot \xi \cdot t_o$; hence, $\mathsf{load}(\tau) \leq m_o \cdot \xi$. ■

Lemma 7.4 above specifies necessary conditions for our partitioning algorithm to successfully partition a sporadic task system; Theorem 7.5 below specifies a *sufficient* condition. But first, a technical lemma that will be used in the proof of Theorem 7.5.

Lemma 7.5 *Suppose that our partitioning algorithm is attempting to schedule task system τ on a platform comprised of unit-capacity processors.*

C1: *If* $\mathsf{system\text{-}util}(\tau) \leq 1$, *then Condition 7.10 is always satisfied.*

C2: *If* $\mathsf{load}(\tau) \leq \frac{1}{2}$, *then Condition 7.9 is always satisfied.*

Proof: The proof of C1 is straightforward, since violating Condition 7.10 requires that $(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_j)$ exceed 1.

To see why C2 holds as well, observe that $\mathsf{load}(\tau) \leq \frac{1}{2}$ implies that $\sum_{\tau_j \in \tau} \mathrm{DBF}(\tau_j, t_o) \leq \frac{t_o}{2}$ for all $t_o \geq 0$. By Inequality 7.2, this in turn implies that $\sum_{\tau_j \in \tau} \mathrm{DBF}^*(\tau_j, t_o) \leq t_o$ for all $t_o \geq 0$; specifically, at $t_o = d_i$ when evaluating Condition 7.9. But, violating Condition 7.9 requires that $(\mathrm{DBF}^*(\tau_i, d_i) + \sum_{\tau_j \in \tau(\pi_k)} \mathrm{DBF}^*(\tau_j, d_i))$ exceed d_i. ■

Corollary 7.1

1. *Any sporadic task system* τ *satisfying* ($\mathsf{system\text{-}util}(\tau) \leq 1 \bigwedge \mathsf{load}(\tau) \leq \frac{1}{2}$) *is successfully partitioned on any number of processors* ≥ 1.

2. *Any constrained sporadic task system* τ *satisfying* ($\mathsf{load}(\tau) \leq \frac{1}{2}$) *is successfully partitioned on any number of processors* ≥ 1.

Proof Sketch: Part 1 follows directly from Lemma 7.5, since Line 3 of the partitioning algorithm in Figure 7.4 will always evaluate to "true."

Part 2 follows from Lemmas 7.5 and 7.3. By Lemma 7.3, we need only determine that Condition 7.9 is satisfied, in order to ensure that Line 3 of the partitioning algorithm in Figure 7.4 evaluate to "true." By Condition C2 of Lemma 7.5, this is ensured by having $\mathsf{load}(\tau) \leq \frac{1}{2}$. ■

Thus, any sporadic task system satisfying both $\mathsf{system\text{-}util}(\tau) \leq 1$ and $\mathsf{load}(\tau) \leq \frac{1}{2}$ is successfully scheduled by our algorithm. We now describe, in Theorem 7.5, what happens when one or both these conditions are not satisfied.

Lemma 7.6 *Let* m_1 *denote the number of processors,* $0 \leq m_1 \leq m$*, on which Condition 7.9 fails when the partitioning algorithm is attempting to map task* τ_i*. It must be the case that*

$$m_1 < \frac{2\mathsf{load}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}}. \tag{7.14}$$

Proof: Since τ_i fails the test of Condition 7.9 on each of the m_1 processors, it must be the case that each such processor π_ℓ satisfies

$$\sum_{\tau_j \in \tau(\pi_\ell)} \mathrm{DBF}^*(\tau_j, d_i) > (d_i - e_i).$$

Summing over all m_1 such processors and noting that the tasks on these processors

is a subset of the tasks in τ, we obtain

$$\begin{aligned}
&\sum_{j=1}^{n} \text{DBF}^*(\tau_j, d_i) > m_1(d_i - e_i) + e_i \\
\Rightarrow \quad & \text{(By Inequality 7.2)} \\
& 2\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) > m_1(d_i - e_i) + e_i \\
\Rightarrow \quad & \frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} > \frac{m_1}{2}(1 - \frac{e_i}{d_i}) + \frac{e_i}{2d_i}. \qquad (7.15)
\end{aligned}$$

By definition of $\mathsf{load}(\tau)$ for sporadic task systems,

$$\frac{\sum_{j=1}^{n} \text{DBF}(\tau_j, d_i)}{d_i} \leq \mathsf{load}(\tau). \qquad (7.16)$$

Chaining Inequalities 7.15 and 7.16 above, we obtain

$$\begin{aligned}
&\frac{m_1}{2}(1 - \frac{e_i}{d_i}) + \frac{e_i}{2d_i} < \mathsf{load}(\tau) \\
\Rightarrow \quad & m_1 < \frac{2\mathsf{load}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}},
\end{aligned}$$

which is as claimed by the lemma. ■

Lemma 7.7 *Let m_2 denote the number of processors, $0 \leq m_2 \leq m - m_1$, on which Condition 7.10 fails (but Condition 7.9 is satisfied) when the partitioning algorithm is attempting to map task τ_i. It must be the case that*

$$m_2 < \frac{\mathsf{system\text{-}util}(\tau) - u_i}{1 - u_i}. \qquad (7.17)$$

Proof: Since none of the m_2 processors satisfies Condition 7.10 for task τ_i, it must be the case that there is not enough remaining utilization on each such processor to accommodate the utilization of task τ_i. Therefore, strictly more than $(1 - u_i)$ of the

capacity of each such processor has already been consumed; summing over all m_2 processors and noting that the tasks already assigned to these processors is a subset of the tasks in τ, we obtain the following upper bound on the value of m_2:

$$\begin{aligned} (1 - u_i)m_2 + u_i &< \sum_{j=1}^{n} u_j \\ \Rightarrow \qquad m_2 &< \frac{\mathsf{system\text{-}util}(\tau) - u_i}{1 - u_i}, \end{aligned}$$

which is as asserted by the lemma. ■

Theorem 7.4 *Any constrained sporadic task system τ is successfully scheduled by our algorithm on m unit-capacity processors, for any m satisfying*

$$m \geq \frac{2\mathsf{load}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)}. \tag{7.18}$$

Proof: Our proof is by contradiction – we will assume that our algorithm fails to partition task system τ on m processors, and prove that, in order for this to be possible, m must violate Inequality 7.18 above. Accordingly, let us suppose that our partitioning algorithm fails to obtain a partition for τ on m unit-capacity processors. In particular, let us suppose that task τ_i cannot be mapped on to any processor. By Lemma 7.3, it must be the case that Condition 7.9 fails for task τ_i on each of the m processors; i.e., m_1 in the statement of Lemma 7.6 is equal to the total number of processors m. Consequently, Inequality 7.14 of Lemma 7.6 yields

$$m < \frac{2\mathsf{load}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} .$$

By Corollary 7.1, it is necessary that $\mathsf{load}(\tau) > \frac{1}{2}$ hold. Since $\mathsf{max\text{-}job\text{-}density}(\tau) \leq 1$ (if not, the system is trivially non-feasible), the right-hand side of the above inequality

is maximized when $\frac{e_i}{d_i}$ is as large as possible, this implies that

$$m < \frac{2\mathsf{load}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)} \ ,$$

which contradicts Inequality 7.18 above. ■

Theorem 7.5 *Any sporadic task system τ is successfully scheduled by our algorithm on m unit-capacity processors, for any m satisfying*

$$m \geq \left(\frac{2\mathsf{load}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)} + \frac{\mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}util}(\tau)}{1 - \mathsf{max\text{-}util}(\tau)} \right). \qquad (7.19)$$

Proof: Once again, our proof is by contradiction – we will assume that our algorithm fails to partition task system τ on m processors, and prove that, in order for this to be possible, m must violate Inequality 7.19 above.

Let us suppose that our partitioning algorithm fails to obtain a partition for τ on m unit-capacity processors. In particular, let us suppose that task τ_i cannot be mapped on to any processor. There are two cases we consider: the case where Condition 7.9 fails and the case where Condition 7.9 is satisfied (implying that Condition 7.10 must have failed). Let Π_1 denote the m_1 processors upon which this mapping fails because Condition 7.9 is not satisfied (hence for the remaining $m_2 \stackrel{\text{def}}{=} (m - m_1)$ processors, denoted Π_2, Condition 7.9 is satisfied but Condition 7.10 is not).

By Lemma 7.5 above, m_1 will equal 0 if $\mathsf{load}(\tau) \leq \frac{1}{2}$, while m_2 will equal 0 if $\mathsf{system\text{-}util}(\tau) \leq 1$. Since we are assuming that the partitioning fails, it is not possible that both $\mathsf{load}(\tau) \leq \frac{1}{2}$ and $\mathsf{system\text{-}util}(\tau) \leq 1$ hold.

Let us extend previous notation as follows: for any collection of processors Π_x, let $\tau(\Pi_x)$ denote the tasks from among $\tau_1, \ldots, \tau_{i-1}$ that have already been allocated to some processor in the collection Π_x. Lemmas 7.6 and 7.7 provide, for task systems on which our partitioning algorithm fails, upper bounds on the values of m_1 (i.e., the

number of processors in Π_1) and m_2 (the number of processors in Π_2) in terms of the parameters of the task system τ.

We now consider three separate cases.

§ **Case (i): ($\mathsf{load}(\tau) > \frac{1}{2}$ and $\mathsf{system\text{-}util}(\tau) \leq 1$).** As stated in Lemma 7.5 (C1), Condition 7.10 is never violated in this case, and m_2 is consequently equal to zero. From this and Lemma 7.6 (Inequality 7.14), we therefore have

$$m < \frac{2\mathsf{load}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}}.$$

In order for our algorithm to successfully schedule τ on m processors, it is sufficient that the negation of the above hold:

$$m \geq \frac{2\mathsf{load}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}}.$$

Since the right-hand side of the above inequality is maximized when $\frac{e_i}{d_i}$ is as large as possible, this implies that

$$m \geq \frac{2\mathsf{load}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)},$$

which certainly holds for any m satisfying the statement of the theorem (Inequality 7.19).

§ **Case (ii): ($\mathsf{load}(\tau) \leq \frac{1}{2}$ and $\mathsf{system\text{-}util}(\tau) > 1$).** As stated in Lemma 7.5 (C2), Condition 7.9 is never violated in this case, and m_1 is consequently equal to zero. From this and Lemma 7.7 (Inequality 7.17), we therefore have

$$m < \frac{\mathsf{system\text{-}util}(\tau) - u_i}{1 - u_i}.$$

We once again observe that it is sufficient that the negation of the above hold in order

for our algorithm to successfully schedule τ on m processors:

$$m \geq \frac{\mathsf{system\text{-}util}(\tau) - u_i}{1 - u_i}.$$

Since the right-hand side of the above inequality is maximized when u_i is as large as possible, this implies that

$$m \geq \frac{\mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}util}(\tau)}{1 - \mathsf{max\text{-}util}(\tau)},$$

which once again holds for any m satisfying the statement of the theorem (Inequality 7.19).

§ **Case (iii): ($\mathsf{system\text{-}util}(\tau) > 1$ and $\mathsf{load}(\tau) > \frac{1}{2}$).** In this case, both m_1 and m_2 may be non-zero. From $m_1 + m_2 = m$ and Inequality 7.17, we may conclude that

$$m_1 > m - \frac{\mathsf{system\text{-}util}(\tau) - u_i}{1 - u_i}. \tag{7.20}$$

For Inequalities 7.20 and 7.14 to both be satisfied, we must have

$$\begin{aligned} & \frac{2\mathsf{load}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} > m - \frac{\mathsf{system\text{-}util}(\tau) - u_i}{1 - u_i} \\ \Rightarrow \quad & m < \frac{2\mathsf{load}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} + \frac{\mathsf{system\text{-}util}(\tau) - u_i}{1 - u_i}. \end{aligned} \tag{7.21}$$

Hence for our algorithm to successfully schedule τ, it is sufficient that the negation of Inequality 7.21 hold:

$$m \geq \left(\frac{2\mathsf{load}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} + \frac{\mathsf{system\text{-}util}(\tau) - u_i}{1 - u_i} \right). \tag{7.22}$$

Observe that the right hand-side of Inequality 7.22 is maximized when u_i and $\frac{e_i}{d_i}$ are both as large as possible; by Equations 2.4 and 3.2, these are defined to be $\mathsf{max\text{-}util}(\tau)$

and $\mathsf{max\text{-}job\text{-}density}(\tau)$ respectively. We hence get Inequality 7.19:

$$m \geq \left(\frac{2\mathsf{load}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)} + \frac{\mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}util}(\tau)}{1 - \mathsf{max\text{-}util}(\tau)} \right)$$

as a sufficient condition for τ to be successfully scheduled by our algorithm. ■

Recall that in the technique of resource augmentation, the performance of the algorithm being discussed is compared with that of a hypothetical optimal one, under the assumption that the algorithm under discussion has access to *more resources* than the optimal algorithm. Using Theorem 7.5 above, we now present such a result concerning our partitioning algorithm.

Theorem 7.6 *Our algorithm makes the following performance guarantees:*

1. *if a constrained sporadic task system is feasible on m_o identical processors each of a particular computing capacity, then our algorithm will successfully partition this system upon a platform comprised of m processors that are each $(2\frac{m_o}{m} + 1 - \frac{1}{m})$ times as fast as the original.*

2. *if an arbitrary sporadic task system is feasible on m_o identical processors each of a particular computing capacity, then our algorithm will successfully partition this system upon a platform comprised of m processors that are each $(3\frac{m_o}{m} + 1 - \frac{2}{m})$ times as fast as the original.*

Proof: Let us assume that $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ is feasible on m_o processors each of computing capacity equal to ξ. Below, we consider separately the cases when τ is a constrained sporadic task system and an arbitrary sporadic task system:

§ 1. τ is a constrained sporadic task system. Since τ is feasible on m_o ξ-speed processors, it follows from Lemma 7.4 that the tasks in τ satisfy the following properties:

$$\mathsf{max\text{-}job\text{-}density}(\tau) \leq \xi, \quad \text{and} \quad \mathsf{load}(\tau) \leq m_o \cdot \xi.$$

Suppose that τ is successfully scheduled by our algorithm on m unit-capacity processors. By substituting the inequalities above in Equation 7.18 of Theorem 7.4, we get

$$\begin{aligned} m &\geq \frac{2\mathsf{load}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)} \\ \Leftarrow \quad m &\geq \frac{2m_o\xi - \xi}{1 - \xi} \\ \equiv \quad \xi &\leq \frac{m}{2m_o + m - 1} \\ \equiv \quad \frac{1}{\xi} &\geq 2\frac{m_o}{m} + 1 - \frac{1}{m}, \end{aligned}$$

which is as claimed in the statement of the theorem.

§ 2. τ is an arbitrary sporadic task system. Since τ is feasible on m_o ξ-speed processors, it follows from Lemma 7.4 that the tasks in τ satisfy the following properties:

$$\mathsf{max\text{-}job\text{-}density}(\tau) \leq \xi, \ \ \mathsf{max\text{-}util}(\tau) \leq \xi, \ \ \mathsf{load}(\tau) \leq m_o{\cdot}\xi, \ \ \text{and} \ \ \mathsf{system\text{-}util}(\tau) \leq m_o{\cdot}\xi.$$

Suppose once again that τ is successfully scheduled by our algorithm on m unit-capacity processors. By substituting the inequalities above in Equation 7.19 of Theorem 7.5, we get

$$\begin{aligned} m &\geq \frac{2\mathsf{load}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)} + \frac{\mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}util}(\tau)}{1 - \mathsf{max\text{-}util}(\tau)} \\ \Leftarrow \quad m &\geq \frac{2m_o\xi - \xi}{1 - \xi} + \frac{m_o\xi - \xi}{1 - \xi} \\ \equiv \quad \xi &\leq \frac{m}{3m_o + m - 2} \\ \equiv \quad \frac{1}{\xi} &\geq 3\frac{m_o}{m} + 1 - \frac{2}{m}, \end{aligned}$$

which is as claimed in the statement of the theorem. ■

By setting $m_o \leftarrow m$ in the statement of Theorem 7.6 above, we immediately have the following corollary.

Corollary 7.2 *Our algorithm makes the following performance guarantees:*

1. *if a constrained sporadic task system is feasible on m identical processors each of a particular computing capacity, then our algorithm will successfully partition this system upon a platform comprised of m processors that are each $(3 - \frac{1}{m})$ times as fast as the original.*

2. *if an arbitrary sporadic task system is feasible on m identical processors each of a particular computing capacity, then our algorithm will successfully partition this system upon a platform comprised of m processors that are each $(4 - \frac{2}{m})$ times as fast as the original.*

■

7.1.5 A Pragmatic Improvement

We have made several approximations in deriving the results above. One of these has been the use of the approximation $\text{DBF}^*(\tau_i, t)$ of Equation 7.1 in Condition 7.9, to determine whether (the first job of) task τ_i can be accommodated on a processor π_k. We could reduce the amount of inaccuracy introduced here, by refining the approximation: rather than approximating $\text{DBF}(\tau_i, t)$ by a single step followed by a line of slope u_i, we could explicitly have included the first k_i steps, followed by a line of slope u_i (as proposed by Albers and Slomka (Albers and Slomka, 2004)). For the

case $k_i = 2$, such an approximation, denoted $\text{DBF}'(\tau_i, t)$ here, is as follows:

$$\text{DBF}'(\tau_i, t) = \begin{cases} 0, & \text{if } t < d_i \\ e_i, & \text{if } d_i \leq t < d_i + p_i \\ 2e_i + u_i \times (t - (d_i + p_i)), & \text{otherwise.} \end{cases} \tag{7.23}$$

If we were to indeed use an approximation comprised of k_i steps, instead of the single-step approximation DBF^*, in Condition 7.9 in determining whether a processor can accommodate an additional task, we would need to explicitly re-check that the first k_i deadlines of all tasks previously assigned to the processor continue to be met. This is because it is no longer guaranteed that the new deadlines (those of τ_i) will occur *after* the deadlines of previously-assigned tasks, and hence it is possible that adding τ_i to the processor will result in some previously-added task missing one of its deadlines. However, the benefit of using better approximations is a greater likelihood of determining a system feasible; we illustrate by an example.

Example 7.1 Suppose that task $\tau_j = (1, 1, 10)$ has already been assigned to processor π_k when task $\tau_i = (1, 2, 20)$ is being considered. Evaluating Condition 7.9, we have

$$\begin{aligned} & d_i - \text{DBF}^*(\tau_j, 2) \geq e_i \\ \equiv \;& 2 - 0.1 \times (2 + 10 - 1) \geq 1 \\ \equiv \;& 2 - 1.1 \geq 1, \end{aligned}$$

which is false; hence, we determine that τ_i fails the test of Condition 7.9 and cannot be assigned to processor π_k.

However, suppose that we were to instead approximate the demand bound function to two steps rather than one, by using the function DBF' (Equation 7.23 above). We

would need to consider two deadlines for both the new task τ_i *as well as* the previously-assigned task τ_j. The deadlines for τ_i are at time-instants 2 and 22, and for τ_j at time-instants 1 and 11. The demand-bound computations at all four deadlines are shown below.

- At $t = 1$: $\text{DBF}'(\tau_j, 1) + \text{DBF}'(\tau_i, 1) = 1 + 0 = 1$, which is ≤ 1.
- At $t = 2$: $\text{DBF}'(\tau_j, 2) + \text{DBF}'(\tau_i, 2) = 1 + 1 = 2$, which is ≤ 2.
- At $t = 11$: $\text{DBF}'(\tau_j, 11) + \text{DBF}'(\tau_i, 11) = 1 + 2 = 3$, which is ≤ 11.
- At $t = 22$: $\text{DBF}'(\tau_j, 22) + \text{DBF}'(\tau_i, 22) = 2 + 3.1 = 5.1$, which is ≤ 22.

Furthermore, τ_i also passes the test of Condition 7.10, since $u_j + u_i = 0.1 + 0.05$ which is ≤ 1. ■

As this example illustrates, the benefit of using a finer approximation is enhanced feasibility: the feasibility test is less likely to incorrectly declare a feasible system to be infeasible. To analyze the run-time cost of this enhancement, assume that for all $\tau_i, \tau_j \in \tau$, $k_i = k_j$. The cost of this improved performance is run-time complexity: rather than just check Condition 7.9 at d_i for each processor during the assignment of task τ_i, we must check a similar condition on a total of $(i \times k_i)$ deadlines over all m processors (observe that this remains polynomial-time, for constant k_i). Hence in practice, we recommend that the largest value of k_i that results in an acceptable run-time complexity for the algorithm be used.

From a theoretical perspective, we were unable to obtain a significantly better bound than the ones in Theorem 7.5 and Corollary 7.2 by using a finer approximation in this manner.

§ **Discussion.** We reiterate that the results in Corollary 7.2 are *not* intended to be used as feasibility tests to determine whether our algorithm would successfully

schedule a given sporadic task system; rather, these properties provide a quantitative measure of how effective our partitioning algorithm is.

Observe that there are two points in our partitioning algorithm during which errors may be introduced. First, we are approximating a solution to a generalization of the bin-packing problem. Second, we are approximating the demand bound function DBF by the function DBF^*, thereby introducing an additional approximation factor of two (Inequality 7.2). While the first of these sources of errors arises even in the consideration of implicit-deadline systems, the second is unique to the generalization in the task model. Indeed, it can be shown that

> *any implicit-deadline sporadic task system τ that is (global or partitioned) feasible on m identical processors can be partitioned in polynomial time, using our partitioning algorithm, upon m processors that are $(2-\frac{1}{m})$ times as fast as the original system, when* EDF *is used to schedule each processor during run-time.*

Thus, the generalization of the task model costs us a factor of 2 in terms of resource augmentation for arbitrary deadlines, and a factor of less than 2, asymptotically approaching 1.5 as $m \to \infty$, for constrained deadlines.

The purpose of the above discussion on the partitioning of implicit-deadline systems is only intended to identify the sources of the error in the approximation factors for constrained-deadline and arbitrary systems (Corollary 7.2). In practice, a system designer would use the tighter analysis of (Lopez et al., 2000; Lopez et al., 2004) for the partitioning implicit-deadline sporadic tasks systems.

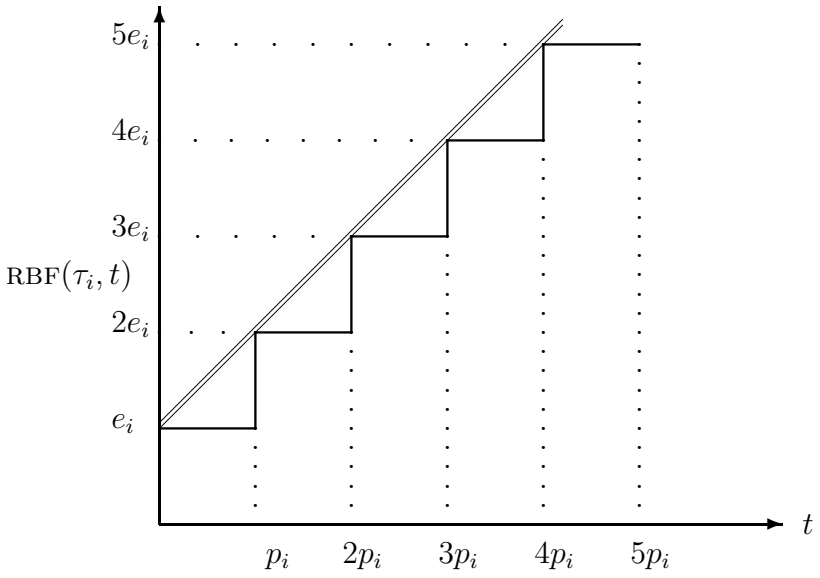


Figure 7.5: A plot of $\text{RBF}(\tau_i, t)$ as a function of t. The double lines indicate the approximation $\text{RBF}^*(\tau_i, t)$.

7.2 DM-based Partitioning

This section focuses on partitioning sporadic task systems when DM is used to schedule each individual processor. In Section 7.2.1, we introduce a different workload characterization called *request-bound function* that is useful in fixed-task-priority systems; in the same section, we define an approximation to the request-bound function and compare it to the demand-bound function. In Section 7.2.2, we present our polynomial-time partitioning algorithm for sporadic task systems and prove its correctness. Section 7.2.3 evaluates the efficacy of the partitioning algorithm in terms of sufficient conditions for success and resource augmentation approximation bounds.

7.2.1 The Request-Bound Function

For any sporadic task τ_i and any real number $t \geq 0$, the *request-bound function* $\text{RBF}(\tau_i, t)$ is the largest cumulative execution requirement of all jobs that can be generated by τ_i to have their arrival times within a contiguous interval of length t.

Every time a task τ_i releases a job, e_i additional units of processor time are requested. The following function provides an upper bound on the total execution time requested by task τ_i at time t (i.e., the scenario where a task releases jobs as soon as legally possible):

$$\text{RBF}(\tau_i, t) \stackrel{\text{def}}{=} \left\lceil \frac{t}{p_i} \right\rceil e_i. \tag{7.24}$$

Fisher and Baruah (Fisher and Baruah, 2005b) proposed a method for approximating RBF; the following function can be obtained by applying the approximation technique:

$$\text{RBF}^*(\tau_i, t) \stackrel{\text{def}}{=} e_i + u_i \times t. \tag{7.25}$$

As stated earlier, it has been shown that the cumulative execution requirement of jobs of τ_i over an interval is maximized if one job arrives at the start of the interval, and subsequent jobs arrive as rapidly as permitted. Intuitively, approximation RBF* (Equation 7.25 above) models this job-arrival sequence by requiring that the first job's deadline be met explicitly by being assigned e_i units of execution upon its arrival, and that τ_i be assigned an additional $u_i \times \Delta t$ of execution over time-interval $[t, t + \Delta t)$, for all instants t after the arrival of the first job, and for arbitrarily small positive Δt. Figure 7.5 illustrates both RBF and RBF*.

§ **Relationship Between** RBF* **and** DBF**.** The following observation will be important in quantitatively evaluating the partitioning algorithm presented in the next section. The next lemma essentially provides an upper bound on $\text{RBF}^*(\tau_i, t)$ in terms of τ_i's utilization and demand-bound function.

Lemma 7.8 *Given a sporadic task τ_i, the following inequality holds for all $t \geq d_i$,*

$$\text{RBF}^*(\tau_i, t) \leq \text{DBF}(\tau_i, t) + (u_i \times t) \tag{7.26}$$

Proof: Observe from the definition of DBF that for $t \geq d_i$, $\text{DBF}(\tau_i, t) \geq e_i$. Substituting this inequality into Equation 7.25, we obtain Equation 7.26. ■

7.2.2 A Polynomial-Time Partitioning Algorithm

Bin-packing heuristics for fixed-task-priority scheduling have been extensively studied (Andersson and Jonsson, 2003; Oh and Baker, 1998). In this section, we are only considering fixed-task-priority scheduling algorithms. Deadline-monotonic scheduling (DM) is known to be optimal for the fixed-task-priority scheduling of constrained sporadic task systems on uniprocessors (Leung and Whitehead, 1982). DM assigns to each task τ_i a priority equivalent to $\frac{1}{d_i}$ and schedules, at any time, the active task with the highest priority. In general, DM performs relatively well in simulations for arbitrary task systems (Baker, 2005b). Therefore, DM is an appropriate algorithm to use to schedule the tasks on each uniprocessor. For the remainder of this section, we will consider a task to fit on a processor if it can be scheduled according to DM with respect to all tasks previously assigned to the processor.

Section 7.2.2.1 presents a partitioning algorithm for sporadic task system where DM is used on each processor. Section 7.2.2.2 shows that this algorithm is correct. Section 7.2.2.3 shows the partitioning algorithm runs in time polynomial in the number of tasks in the task system.

7.2.2.1 Algorithm DM-PARTITION

We now describe a simple partitioning algorithm called DM-PARTITION. Given a sporadic task system τ comprised of n sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$, and a processing platform Π comprised of m unit-capacity processors $\pi_1, \pi_2, \ldots, \pi_m$, DM-PARTITION

will attempt to partition τ among the processors of Π. The DM-PARTITION algorithm is a variant of a bin-packing heuristic known as *first-fit-decreasing*. For this section, we will assume the tasks of τ_i are indexed in non-decreasing order of their relative deadline (i.e., $d_i \leq d_{i+1}$, for $1 \leq i < n$).

The DM-PARTITION algorithm considers the tasks in decreasing DM-priority order (i.e., $\tau_1, \tau_2, \ldots$). We will now describe how to assign task τ_i assuming that tasks $\tau_1, \tau_2, \ldots, \tau_{i-1}$ have already successfully been allocated among the m processors. Let $\tau(\pi_\ell)$ denote the set of tasks already assigned to processor π_ℓ where $1 \leq \ell \leq m$. Considering the processors $\pi_1, \pi_2, \ldots, \pi_m$ in any order, we will assign task τ_i to the first processor π_k, $1 \leq k \leq m$, that satisfies the following two conditions:

$$\left(d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{RBF}^*(\tau_j, d_i) \right) \geq e_i \tag{7.27}$$

and

$$\left(1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \right) \geq u_i. \tag{7.28}$$

If no such π_k exists, then Algorithm DM-PARTITION returns PARTITIONING FAILED: it is unable to conclude that sporadic task system τ is feasible upon the m-processor platform. Otherwise, DM-PARTITION returns PARTITIONING SUCCEEDED.

§ **Constrained Task Systems.** We may eliminate the need for checking Inequality 7.28 by considering constrained task systems. For these systems, it is sufficient to check only Inequality 7.27:

Lemma 7.9 *For a constrained sporadic task system τ, any $\tau_i \in \tau$ and $\pi_k \in \Pi$ satisfying Inequality 7.27, while attempting to assign τ_i to π_k in* DM-PARTITION*, will also satisfy Inequality 7.28.*

Proof: Observe that Inequality 7.27 implies

$$\begin{aligned} & d_i - \textstyle\sum_{\tau_j \in \tau(\pi_k)} (e_j + d_i \times u_j) \geq e_i \\ \Rightarrow \quad & 1 - \left(\textstyle\sum_{\tau_j \in \tau(\pi_k)} \frac{e_j}{d_i} \right) - \left(\textstyle\sum_{\tau_j \in \tau(\pi_k)} u_j \right) \geq \frac{e_i}{d_i}. \end{aligned}$$

Since $d_i \leq p_i$,

$$1 - \sum_{\tau_j \in \tau(\pi_k)} u_j \geq \frac{e_i}{p_i} = u_i.$$

Thus, Inequality 7.28 will evaluate to "true." ■

7.2.2.2 Proof of Correctness

In order to prove that DM-PARTITION is correct, we are obligated to show that after each assignment of a task to a processor, the system is DM-feasible. In particular, we must prove that if DM-PARTITION returned PARTITIONING SUCCEEDED then the set of tasks assigned to each processor is DM-feasible on that processor (Theorem 7.7). However, before we can prove the correctness of DM-PARTITION, we need a feasibility test for uniprocessors that uses RBF*. The following lemma provides such a test.

Lemma 7.10 *Tasks $\tau(\pi_k)$ are DM-feasible on processor π_k if for each $\tau_i \in \tau(\pi_k)$ and $a \in \mathbb{N}^+$ the following condition is satisfied:*

$$\begin{aligned} \exists t : \; & (a-1)p_i < t \leq (a-1)p_i + d_i :: \\ & \left(a e_i + \textstyle\sum_{\substack{\tau_j \in \tau(\pi_k) \\ æ < i}} \mathrm{RBF}^*(\tau_j, t) \leq t \right). \end{aligned} \tag{7.29}$$

Proof: By Lemma 9 of (Fisher and Baruah, 2005a), if Inequality 7.29 is satisfied, then algorithm APPROX$(\tau(\pi_k), 0.5)$ (described in (Fisher and Baruah, 2005a)) will return "$\tau(\pi_k)$ is DM-feasible on π_k." By Theorem 5 of (Fisher and Baruah, 2005a), APPROX is correct, and the lemma follows. ■

The next lemma shows that algorithm DM-PARTITION maintains the invariant that Inequalities 7.27 and 7.28 remain true for all assigned tasks during every step of the algorithm. The invariant is useful to show that the assignment of a task to a processor does not affect the feasibility of the previously-assigned tasks.

Lemma 7.11 *For each $\pi_k \in \Pi$, the following conditions always hold for algorithm* DM-PARTITION*: for every $\tau_j \in \tau(\pi_k)$,*

$$\left(d_j - \sum_{\substack{\tau_\ell \in \tau(\pi_k) \\ \ell < j}} \text{RBF}^*(\tau_\ell, d_j) \right) \geq e_j \tag{7.30}$$

and

$$1 - \sum_{\substack{\tau_\ell \in \tau(\pi_k) \\ \ell < j}} u_\ell \geq u_j. \tag{7.31}$$

Proof: Observe that Inequality 7.30 or 7.31 for some $\pi_k \in \Pi$ can only be falsified by the assignment of a task τ_i by algorithm DM-PARTITION. Thus, we only need to show that Lemma 7.11 is maintained after every task assignment. We will prove this by induction on the assignment of tasks:

Base Case: Prior to any assignment of a task to a processor by DM-PARTITION, each processor is empty. Therefore, Inequalities 7.30 and 7.31 are vacuously true.

Inductive Step: Assume Inequalities 7.30 and 7.31 remain true for the assignments of $\tau_1, \tau_2, \ldots, \tau_{i-1}$ (by the inductive hypothesis). We must show that Inequalities 7.30 and 7.31 continue to hold after the assignment of τ_i. Let τ_i be assigned to processor π_k by DM-PARTITION. Observe that Inequalities 7.30 and 7.31 for $\pi_s \neq \pi_k$ are unaffected and remain true. Additionally, for $\tau_j \in \tau(\pi_k) - \{\tau_i\}$, it must be that $j < i$, since tasks are assigned in order by

DM-PARTITION. Thus, for all $\tau_j \in \tau(\pi_k) - \{\tau_i\}$, Inequalities 7.30 and 7.31 are unaffected as well. The lemma follows as DM-PARTITION ensures that Inequalities 7.30 and 7.31 are true (via Inequalities 7.27 and 7.28) for τ_i and π_k upon assigning τ_i.

■

Finally, we are prepared to prove the correctness of algorithm DM-PARTITION.

Theorem 7.7 *If* DM-PARTITION *returns* PARTITIONING SUCCEEDED*, then the tasks of* τ *assigned to processors of* Π *are* DM*-feasible on their respective processors.*

Proof: The proof is by contradiction. Since DM-PARTITION returned PARTITIONING SUCCEEDED, then each task of τ is assigned to a processor of Π. For the sake of contradiction, assume there exists a processor $\pi_k \in \Pi$ such that each task $\tau(\pi_k)$ will not always meet all deadlines when scheduled on π_k. By Lemma 7.10, this implies that there exists a task $\tau_i \in \tau(\pi_k)$ and $a \in \mathbb{N}^+$ such that

$$\begin{array}{l} \forall t: \ (a-1)p_i < t \leq (a-1)p_i + d_i :: \\ \qquad \left(ae_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \\ j<i}} \text{RBF}^*(\tau_j, t) > t \right). \end{array} \tag{7.32}$$

By Lemma 7.11, each task-processor assignment satisfies Inequalities 7.30 and 7.31. Inequality 7.30 implies

$$e_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \\ j<i}} (e_j + d_i u_j) \leq d_i \ \ \text{(by defititon of RBF}^*\text{)}. \tag{7.33}$$

The next equation follows from multiplying both sides of Inequality 7.31 by $(a-1)p_i$:

$$[(a-1)p_i] \left(u_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \\ j<i}} u_j \right) \leq (a-1)p_i. \tag{7.34}$$

By summing the Inequalities 7.33 and 7.34, we obtain

$$ae_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \\ j<i}} (e_j + u_j \left[(a-1)p_i + d_i\right]) \leq (a-1)p_i + d_i$$

$$\Rightarrow ae_i + \sum_{\substack{\tau_j \in \tau(\pi_k) \\ j<i}} \text{RBF}^*(\tau_j, (a-1)p_i + d_i) \leq (a-1)p_i + d_i.$$

The last inequality contradicts Inequality 7.32. Therefore, our supposition that there exists a π_k where $\tau(\pi_k)$ does not always meet all deadlines is incorrect. It follows that for each $\pi_k \in \Pi$, $\tau(\pi_k)$ is DM-feasible on π_k. ■

7.2.2.3 Computational Complexity

Obviously, to sort each task in (non-decreasing) relative deadline order requires $\Theta(n \lg n)$ time. In attempting to map task τ_i, observe that Algorithm DM-PARTITION essentially evaluates, in Equations 7.27 and 7.28, the workload generated by the previously-mapped $(i-1)$ tasks on each of the m processors. Since $\text{RBF}^*(\tau_j, t)$ can be evaluated in constant time (see Equation 7.25), a straightforward computation of this workload would require $\mathcal{O}(i+m)$ time. Hence the runtime of the algorithm in mapping all n tasks is no more than $\sum_{i=1}^{n} \mathcal{O}(i+m)$, which is $\mathcal{O}(n^2)$ under the reasonable assumption that $m \leq n$.

7.2.3 Theoretical Evaluation

In this section, we quantitatively evaluate the effectiveness of DM-PARTITION by providing sufficient conditions for success (Section 7.2.3.1) and in terms of a resource augmentation approximation bounds (Section 7.2.3.2).

7.2.3.1 Sufficient Schedulability Conditions

The main results of this section (Theorems 7.8 and 7.9) provide an upper-bound on the minimum number of processors necessary for DM-PARTITION to successfully partition a sporadic task system. The bound is dependent on the utilization and load parameter of the task system.

Before presenting the main theorems of this section, we require several technical lemmas. The next lemma characterizes the conditions under which Inequalities 7.27 or 7.28 are trivially satisfied.

Lemma 7.12 *Given a sporadic task system τ and an m unit-capacity processor system Π,* DM-PARTITION *has the following properties.*

P1: *If* $\mathsf{system\text{-}util}(\tau) \leq 1$*, Inequality 7.28 is always satisfied.*

P2: *If* $\mathsf{system\text{-}util}(\tau) \leq 1$ *and* $\mathsf{load}(\tau) \leq 1 - \mathsf{system\text{-}util}(\tau)$*, then Inequality 7.27 is always satisfied.*

Proof: P1 is trivially true, since violating Inequality 7.28 requires that $(u_i + \sum_{\tau_j \in \tau(\pi_k)} u_k)$ exceed 1.

To see P2, observe that $\mathsf{system\text{-}util}(\tau) \leq 1$ and $\mathsf{load}(\tau) \leq 1 - \mathsf{system\text{-}util}(\tau)$ implies for all $t \geq 0$,

$$\begin{aligned} & \frac{\sum_{j=1}^{n} \mathrm{DBF}(\tau_j, t)}{t} \leq 1 - \mathsf{system\text{-}util}(\tau) \\ \Rightarrow \quad & \sum_{j=1}^{n} \mathrm{DBF}(\tau_j, d_i) \leq d_i - d_i \mathsf{system\text{-}util}(\tau) \\ \Rightarrow \quad & \sum_{j=1}^{n} \mathrm{DBF}(\tau_j, d_i) + d_i \mathsf{system\text{-}util}(\tau) \leq d_i. \end{aligned} \tag{7.35}$$

Consider any $\tau_i \in \tau$. Observe that for all $j < i$, $d_j \leq d_i$. Thus, for all $j \leq i$, $\mathrm{DBF}(\tau_j, d_i) \geq e_j$. Observing that $\{\tau_1, \tau_2, \ldots, \tau_{i-1}\} \subset \tau$ and combining lower-bound on

DBF with Inequality 7.35 implies

$$\begin{aligned} & e_i + \sum_{j=1}^{i-1} e_j + d_i \left(\sum_{j=1}^{i-1} u_j \right) \leq d_i \\ \Rightarrow \quad & e_i + \sum_{j=1}^{i-1} (e_j + d_i u_j) \leq d_i \\ \Rightarrow \quad & d_i - \sum_{j=1}^{i-1} \text{RBF}^*(\tau_j, d_i) \geq e_i. \end{aligned} \tag{7.36}$$

P2 follows from Inequality 7.36 and noting that for all $\pi_k \in \Pi$ $\tau(\pi_k) \subseteq \{\tau_1, \tau_2, \ldots, \tau_{i-1}\}$.

■

Corollary 7.3 *Any sporadic task system* τ *with* $(\mathsf{system\text{-}util}(\tau) \leq 1) \wedge (\mathsf{load}(\tau) \leq 1 - \mathsf{system\text{-}util}(\tau))$ *can be successfully partitioned using* DM-PARTITION *on* $m \geq 1$ *processors.*

The next two lemmas provide an upper-bound on the number of processor on which either Inequality 7.27 or 7.28 of DM-PARTITION will evaluate to false.

Lemma 7.13 *Let* m_1 *denote the number of processors on which Inequality 7.27 fails while* DM-PARTITION *attempts to assign* τ_i *to a processor. It must be the case that*

$$m_1 < \frac{\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}}. \tag{7.37}$$

Proof: Let $\Pi_1 \subseteq \Pi$ be the set of all processors on which Inequality 7.27 fails. Then, for all $\pi_k \in \Pi_1$,

$$\begin{aligned} & d_i - \sum_{\tau_j \in \tau(\pi_k)} \text{RBF}^*(\tau_j, d_i) < e_i \\ \Rightarrow \quad & d_i - \sum_{\tau_j \in \tau(\pi_k)} (\text{DBF}(\tau_j, d_i) + d_i u_j) < e_i. \end{aligned} \tag{7.38}$$

Inequality 7.38 follows from Inequality 7.26 of Lemma 7.8. By noting that for each π_k, $\tau(\pi_k)$ is a subset of $\tau - \{\tau_i\}$, and summing Inequality 7.38 over all $\pi_k \in \Pi_1$, we

obtain,

$$\begin{aligned} & m_1 d_i - \sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) - d_i \textsf{system-util}(\tau) < m_1 e_i - e_i \\ \Rightarrow \quad & m_1(d_i - e_i) + e_i < \sum_{j=1}^{n} \text{DBF}(\tau_j, d_i) + d_i \textsf{system-util}(\tau) \\ \Rightarrow \quad & m_1(1 - \tfrac{e_i}{d_i}) + \tfrac{e_i}{d_i} < \textsf{load}(\tau) + \textsf{system-util}(\tau). \end{aligned}$$

The last inequality implies the lemma. ■

Lemma 7.14 *Let m_2 denote the number of processors on which Inequality 7.28 fails and Inequality 7.27 is satisfied while* DM-PARTITION *attempts to assign τ_i to a processor. It must be the case that*

$$m_2 < \frac{\textsf{system-util}(\tau) - u_i}{1 - u_i}. \tag{7.39}$$

Proof: Let $\Pi_2 \subseteq \Pi - \Pi_1$ be the set of all processors on which Inequality 7.28 fails (while Inequality 7.27 is satisfied). Then, for all $\pi_k \in \Pi_2$,

$$1 - \sum_{\tau_j \in \tau(\pi_k)} u_j < u_i.$$

Noting that for each π_k, $\tau(\pi_k)$ is a subset of τ, and summing Inequality 7.2.3.1 over all $\pi_k \in \Pi_1$, we obtain,

$$\begin{aligned} & m_2 - \textsf{system-util}(\tau) < m_2 u_i - u_i \\ \Rightarrow \quad & m_2(1 - u_i) < \textsf{system-util}(\tau) - u_i. \end{aligned}$$

The last inequality implies the lemma. ■

We are now prepared to prove the sufficient conditions for the success of

DM-PARTITION over a set of constrained task systems (Theorem 7.8) and arbitrary task systems (Theorem 7.9).

Theorem 7.8 *Any* constrained *sporadic task system τ is successfully scheduled by* DM-PARTITION *on m unit-capacity processors for any m satisfying*

$$m \geq \frac{\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)}. \tag{7.40}$$

Proof: We will prove the contrapositive of the theorem. Assume that DM-PARTITION fails to assign task τ_i to any processor of Π. By Lemma 7.9, Inequality 7.27 is false for every $\pi_k \in \Pi$ (i.e., $m = m_1$). Thus, by Lemma 7.13,

$$m < \frac{\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}}. \tag{7.41}$$

Corollary 7.3 implies that $\mathsf{system\text{-}util}(\tau) > 1$ or $\mathsf{load}(\tau) > 1 - \mathsf{system\text{-}util}(\tau)$. Both inequalities imply that $\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) > 1$. Therefore, the right-hand-side of Inequality 7.41 is maximized when $\frac{e_i}{d_i}$ is as large as possible. It follows that

$$m < \frac{\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)},$$

which proves the contrapositive of the theorem. ■

Theorem 7.9 *Any sporadic task system τ is successfully scheduled by* DM-PARTITION *on m unit-capacity processors for any m satisfying*

$$m \geq \frac{\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)} + \frac{\mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}util}(\tau)}{1 - \mathsf{max\text{-}util}(\tau)}. \tag{7.42}$$

Proof: We will prove the contrapositive of the theorem. Assume that DM-PARTITION fails to assign task τ_i to any processor of Π. We now consider four subcases based on

the values of $\mathsf{system\text{-}util}(\tau)$ and $\mathsf{load}(\tau)$. Each of the subcases, is either not possible or will imply the contrapositive of the theorem.

1. $\mathsf{system\text{-}util}(\tau) \leq 1 \wedge \mathsf{load}(\tau) \leq 1 - \mathsf{system\text{-}util}(\tau)$: By Corollary 7.3, τ would be trivially partitionable on a single processor by DM-PARTITION. Therefore, this subcase is impossible as it contradicts our supposition that DM-PARTITION fails to assign some task a processor.

2. $\mathsf{system\text{-}util}(\tau) \leq 1 \wedge \mathsf{load}(\tau) > 1 - \mathsf{system\text{-}util}(\tau)$: By Lemma 7.12, Inequality 7.28 is always satisfied. Therefore, Inequality 7.27 must be violated for all π_k when attempting to assign τ_i (i.e., $m = m_1$). Using reasoning identical to Theorem 7.8, we will obtain

$$m < \frac{\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)},$$

   from which the contrapositive of the theorem follows.

3. $\mathsf{system\text{-}util}(\tau) > 1 \wedge \mathsf{load}(\tau) \leq 1 - \mathsf{system\text{-}util}(\tau)$: Notice that $\mathsf{system\text{-}util}(\tau) > 1$ implies that $\mathsf{load}(\tau) < 0$. This subcase is trivially impossible.

4. $\mathsf{system\text{-}util}(\tau) > 1 \wedge \mathsf{load}(\tau) > 1 - \mathsf{system\text{-}util}(\tau)$: Recall that m_1 denotes the number of processor on which Inequality 7.27 of DM-PARTITION fails while attempting to assign τ_i. m_2 denotes the remaining processors on which Inequality 7.28 fails. Therefore, $m = m_1 + m_2$. From Lemma 7.14,

$$m_2 < \frac{\mathsf{system\text{-}util}(\tau) - u_i}{1 - u_i}. \tag{7.43}$$

   Lemma 7.13 implies

$$m_1 < \frac{\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}}. \tag{7.44}$$

Summing Inequalities 7.44 and 7.43, we obtain

$$m = m_1 + m_2 < \frac{\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) - \frac{e_i}{d_i}}{1 - \frac{e_i}{d_i}} + \frac{\mathsf{system\text{-}util}(\tau) - u_i}{1 - u_i}. \tag{7.45}$$

Since $\mathsf{system\text{-}util}(\tau) > 1$ and $\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) > 1$, the right-hand-side of Inequality 7.45 is maximized when both $\frac{e_i}{d_i}$ and u_i are as large as possible. This implies the contrapositive of the theorem.

■

7.2.3.2 Resource Augmentation

Theorem 7.10 *Given an identical multiprocessor platform* Π *with* m *processors and a sporadic task system* τ *(global or partition) feasible on* Π *, the* DM-PARTITION *algorithm has the following performance guarantees:*

1. *if* τ *is a* constrained *system, then* DM-PARTITION *will successfully partition* τ *upon a platform comprised of* m *processors that are each* $\left(3 - \frac{1}{m}\right)$ *times as fast as the processors of* Π*.*

2. *if* τ *is an* arbitrary *system, then* DM-PARTITION *will successfully partition* τ *upon a platform comprised of* m *processors that are each* $\left(4 - \frac{2}{m}\right)$ *times as fast as the processors of* Π*.*

Proof: Assume that we are given task system τ feasible on m processors each of speed ξ, it follows from Lemma 7.4 that τ must satisfy the following properties:

$$\begin{aligned} &\mathsf{load}(\tau) \leq m \cdot \xi && \mathsf{system\text{-}util}(\tau) \leq m \cdot \xi \\ &\mathsf{max\text{-}job\text{-}density}(\tau) \leq \xi && \mathsf{max\text{-}util}(\tau) \leq \xi. \end{aligned} \tag{7.46}$$

Suppose that τ is successfully scheduled on m unit-capacity processor by DM-PARTITION.

We first show (1) by substituting the Inequalities of 7.46 above into Inequality 7.40 of Theorem 7.8:

$$\begin{aligned} m \geq \frac{\mathsf{load}(\tau)+\mathsf{system\text{-}util}(\tau)-\mathsf{max\text{-}job\text{-}density}(\tau)}{1-\mathsf{max\text{-}job\text{-}density}(\tau)} \\ \Leftarrow \quad m \geq \tfrac{m\xi+m\xi-\xi}{1-\xi} \\ \Leftarrow \quad m \geq \tfrac{2m\xi-\xi}{1-\xi} \\ \equiv \quad \xi \leq \tfrac{m}{3m-1} \\ \equiv \quad \tfrac{1}{\xi} \geq 3 - \tfrac{1}{m}. \end{aligned}$$

The last implication is claimed by (1) of the theorem.

To show (2), we similarly substitute the Inequalities of 7.46 into Inequality 7.42 of Theorem 7.9:

$$\begin{aligned} m \geq \frac{\mathsf{load}(\tau)+\mathsf{system\text{-}util}(\tau)-\mathsf{max\text{-}job\text{-}density}(\tau)}{1-\mathsf{max\text{-}job\text{-}density}(\tau)} + \frac{\mathsf{system\text{-}util}(\tau)-\mathsf{max\text{-}util}(\tau)}{1-\mathsf{max\text{-}util}(\tau)} \\ \Leftarrow \quad m \geq \tfrac{2m\xi-\xi}{1-\xi} + \tfrac{m\xi-\xi}{1-\xi} \\ \equiv \quad \xi \leq \tfrac{m}{4m-2} \\ \equiv \quad \tfrac{1}{\xi} \geq 4 - \tfrac{2}{m}, \end{aligned}$$

which is claimed by (2) of the theorem. ■

7.2.3.3 Comparison with Prior Implicit-Deadline Partitioning Results

Polynomial-time partitioning algorithms for implicit-deadline systems are known (Oh and Baker, 1998; Andersson and Jonsson, 2003). To evaluate the theoretical loss of schedulability resulting from the move to a more general task model, let us consider the resource augmentation bound of the partitioning algorithm analyzed by Andersson and Jonsson (Andersson and Jonsson, 2003). By using the logic of Theorem 7.10, the following theorem can be obtained from their $0.5m$ utilization bound:

Theorem 7.11 *Given an identical multiprocessor platform* Π *with* m *processors and an implicit-deadline sporadic task system* τ *(global or partition) feasible on* Π*, can be partitioned in polynomial-time upon a platform comprised of* m *processors that are each approximately 2 times as fast as the processors of* Π*.*

7.3 Summary

Most prior theoretical research concerning the multiprocessor partitioning of sporadic task systems has imposed the additional constraint that all tasks have their deadline parameter equal to their period parameter. In this chapter, we have removed this constraint, and have considered the partitioning of arbitrary sporadic task systems upon preemptive multiprocessor platforms. We have designed an algorithm for performing the partitioning of a given collection of sporadic tasks upon a specified number of processors, and have proved the correctness of, and evaluated the effectiveness of, this partitioned algorithm. The techniques developed in this chapter allow for partitioning to be achieved in polynomial-time while still ensuring a low constant-factor resource-augmentation approximation ratio.

Chapter 8

Conclusions and Future Work

The increasing ubiquity of real-time systems has led to a diverse range in the behavior and complexity of real-time applications. Researchers have addressed the increased diversity for uniprocessor systems by developing general real-time task models that successfully characterize complex interactions by real-time applications. With the recent emergence and commercial acceptance of multicore platforms, many future real-time systems will undoubtedly be multiprocessor systems. However, until recently, most multiprocessor real-time research has focused primarily upon simple task models such as the periodic or LL task model; the techniques available to a real-time system designer to temporally verify the correctness of an application on multiprocess exhibiting more complex behavior have been limited. The results in this dissertation increase the types of real-time applications and behavior that can be temporally verified upon a multiprocessor platform.

Towards this the goal of verifying increasingly general task systems upon multiprocessor platforms, we have proposed the following thesis: *Optimal online multiprocessor real-time scheduling algorithms for sporadic and more general task systems are impossible; however, efficient, online scheduling algorithms and associated feasibility and schedulability tests, with provably bounded deviation from any optimal test, exist.*

In Section 8.1, we summarize the contributions of this dissertation that support this thesis. In Section 8.2, we describe related work by us not included in this dissertation. Section 8.3 describes a future research agenda on problems arising from this dissertation and other related problems. We conclude with some remarks in Section 8.4.

8.1 Summary of Results

We will now summarize the contributions of this dissertation.

8.1.1 Efficient Workload Characterization for General Task Systems

In Chapter 2, we observed that many traditional real-time workload characterizations that were effective in validation techniques for LL multiprocessor systems perform arbitrarily poorly for analyzing sporadic or more general task systems. Chapter 3 present a characterization of real-time work load using a combination of demand-based load and maximum job density (represented by load and max-job-density, respectively). We describe how these two characterizations can be effectively determined for all partially-specified recurrent task systems that generalize the sporadic task model and those satisfying the task independence assumptions. To support this claim, we develop several algorithms to efficiently compute load for sporadic task systems. We develop two algorithms that approximate load to within an additive ϵ of its actual value in pseudo-polynomial time and a PTAS that requires $\mathcal{O}(n^3/\epsilon)$.

8.1.2 Multiprocessor Feasibility Tests

In Chapter 4, we develop feasibility tests for real-time instances based on load and max-job-density. We develop a test for the full-migration feasibility of a real-time in-

Scheduling Paradigm:	Task Model	
	Sporadic	**GMF/Recurring/etc..**
Full-Migration	$\sqrt{2}+1$ (Theorem 4.4)	
Restricted-Migration	Constrained: $3-\frac{1}{m}$	$4-\frac{1}{m}$ (Theorem 4.2)
Partitioned	Arbitrary: $4-\frac{2}{m}$ (Corollary 7.2)	Future Work

Table 8.1: A summary of the resource-augmentation approximation ratios guaranteed by the feasibility tests presented in this dissertation. In each entry, we give the approximation ratio and the corresponding theorem number. The results for the partitioned feasibility correspond to the schedulability tests of Chapter 7 (i.e., a schedulability test is also by definition a feasibility test). Also, the results for partitioned feasibility of sporadic task systems immediately imply resource-augmentation results for restricted-migration feasibility of sporadic task systems since a partitioned schedule is also by definition a restricted-migration schedule.

stance; the test obtain is $\mathsf{load}(I) \leq \frac{m-(m-2)\cdot\mathsf{max\text{-}job\text{-}density}(I)}{1+\mathsf{max\text{-}job\text{-}density}(I)}$. We show that this test is a factor of at most ≈ 1.61 away from the optimal test that could be obtained using load and max-job-density. We also develop a test for restricted-migration feasibility: $\mathsf{load}(I) \leq \frac{m-(m-1)\cdot\mathsf{max\text{-}job\text{-}density}(I)}{3}$. We show that this test is a factor of at most $\frac{8}{3}$ from the optimal attainable test using load and max-job-density. Per the discussion of Chapter 3 in Section 3.3, these tests may be immediately used as feasibility test for partially-specified recurrent task systems where load and max-job-density parameter may be determined.

For partitioned systems, the schedulability tests of Chapter 7 give feasibility tests for partitioned sporadic task systems. The resource augmentation guarantees for the feasibility tests obtained in this dissertation are summarized in Table 8.1.

8.1.3 Impossibility of Optimal Online Multiprocessor Scheduling

The feasibility tests of Chapter 4 imply that for a task system satisfying there is a valid interleaving of jobs' execution upon an m-processor platform that would meet all deadlines. However, Chapter 5 shows that even though there may exist a valid schedule there may not be an online algorithm that always generates a schedule meeting all the task system's deadlines. To show this, we give an example sporadic task system that is feasible upon two processors (the proof of feasibility is contained in Appendix A). We show for this task system, that there exists a sequence of job releases that causes any choice made by an online scheduling algorithm to result in a deadline miss. The existence of such a feasible task system implies that optimal online multiprocessor scheduling algorithms do not exist for sporadic or more general tasks systems. Thus, the search for scheduling algorithms and validation techniques with constant-factor approximation ratios is justified.

8.1.4 Multiprocessor Schedulability Tests

We obtained schedulability tests for scheduling algorithms in Chapters 6 and 7. In Chapter 6, we obtained schedulability tests for DM and EDF under the full-migration scheduling paradigm. For DM, if $\mathsf{load}(I) \leq \frac{m-(m-1)\cdot\mathsf{max\text{-}job\text{-}density}(I)}{3}$ is satisfied then DM will meet all deadlines when scheduling I upon an m-processor platform. For EDF, if both $\mathsf{load}(I) \leq \frac{(\sqrt{2}-1)m^2}{2m-1}$ and $\mathsf{max\text{-}job\text{-}density}(I) \leq \frac{(\sqrt{2}-1)m}{2m-1}$ is satisfied, then I may be scheduled by EDF to meet all deadlines upon an m-processor platform.

In Chapter 7, we obtained schedulability tests for sporadic task systems assuming either EDF or DM is used to schedule each individual processor. Furthermore, we obtained conditions for arbitrary sporadic task systems (i.e., no restriction place

<table>
<tr><th rowspan="2">Scheduling Paradigm:</th><th colspan="2">Task Model</th></tr>
<tr><th>Sporadic</th><th>GMF/Recurring/etc..</th></tr>
<tr><td>Full-Migration</td><td colspan="2">DM: $4 - \frac{1}{m}$ (Theorem 6.2)
EDF: $2 + 2\sqrt{2} - \frac{\sqrt{2}+1}{m}$ (Theorem 6.4)</td></tr>
<tr><td>Restricted-Migration</td><td rowspan="2">For both EDF and DM
Constrained: $3 - \frac{1}{m}$
Arbitrary: $4 - \frac{2}{m}$
(Corollary 7.2)</td><td rowspan="2">Future Work</td></tr>
<tr><td>Partitioned</td></tr>
</table>

Table 8.2: A summary of the resource-augmentation approximation ratios guaranteed by the schedulability tests presented in this dissertation. In each entry, we give the approximation ratio and the corresponding theorem number.

relationship between d_i and p_i for any task τ_i) and tighter conditions for constrained-deadline sporadic task systems (i.e., task systems where each task τ_i has $d_i \leq p_i$). For partitioned platforms scheduled by EDF on each uniprocessor, an arbitrary sporadic task system τ is partitionable according to Algorithm PARTITION if $m \geq \frac{2 \cdot \mathsf{load}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)} + \frac{\mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}util}(\tau)}{1 - \mathsf{max\text{-}util}(\tau)}$, and constrained-deadline system τ is partitionable if $m \geq \frac{2 \cdot \mathsf{load}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)}$. For partitioned platforms scheduled by DM on each processor, an arbitrary sporadic task system τ is partitionable according to DM-PARTITION if $m \geq \frac{\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)} + \frac{\mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}util}(\tau)}{1 - \mathsf{max\text{-}util}(\tau)}$, and constrained-deadline system τ is partitionable if $m \geq \frac{\mathsf{load}(\tau) + \mathsf{system\text{-}util}(\tau) - \mathsf{max\text{-}job\text{-}density}(\tau)}{1 - \mathsf{max\text{-}job\text{-}density}(\tau)}$. The resource augmentation guarantees for the schedulability tests obtained in this dissertation are summarized in Table 8.2.

8.2 Related Research Contributions

We now outline our other research contributions not previously mentioned in this dissertation.

§ **Partitioning of real-time task with memory constraints.** Most prior theoretical research on partitioning algorithms for real-time multiprocessor platforms has focused on ensuring that the cumulative computing requirements of the tasks assigned to each processor do not exceed the processor's processing power. However, many multiprocessor platforms have only limited amounts of local per-processor memory (e.g., Intel's IXP2XXX series of network processors); if the memory limitation of a processor is not respected, thrashing between "main" memory and the processor's local memory may occur during run-time and may result in performance degradation. Our research has developed an approximation algorithm for task partitioning (Fisher et al., 2005). In addition, we have considered an approximation algorithm (Baruah and Fisher, 2005a) for architectures that allow instruction-memory to be compressed at the expense of additional instruction decoding time (examples include the ARM Thumb and MIPS16); the goal in partitioning such an architecture is to find a compression that minimizes the cumulative code-size of each task while simultaneously ensuring temporal constraints.

§ **Resource-locking durations in uniprocessor systems.** There has recently been much interest in the design and implementation of *open* environments (Deng and Liu, 1997) (also called *hierarchically-scheduled systems* or real-time *virtualization*) for real-time applications. Such open environments allow for multiple independently developed and validated real-time applications to co-execute upon a single shared platform. Given the specifications of such a system, an important objective is to determine, for each non-preemptable serially reusable resource, the length of the longest interval of time for which the resource may be locked. In (Bertogna et al., 2006; Fisher et al., 2007a), we have extended current scheduling-theoretic analysis techniques to obtain resource-locking durations. A high-level scheduler that arbitrates access to such non-preemptable shared resources among different applications

will use this knowledge to determine the schedulability of the entire system.

§ **Fully polynomial-time approximation algorithms for uniprocessor schedulability analysis.** In *static-priority* scheduling algorithms for sporadic task systems, each task is assigned a distinct priority, and all jobs of a task execute at the task's priority. The computational complexity of analyzing whether a given static-priority assignment is feasible on a single processor is currently unknown; the best known tests are pseudo-polynomial. In (Fisher and Baruah, 2005b; Fisher and Baruah, 2005a), we propose a fully polynomial-time approximation (FPTAS) algorithm with the following guarantee: for any specified value of ϵ, where $0 < \epsilon < 1$, the FPTAS correctly identifies, in time polynomial in the number of tasks in the task system and $1/\epsilon$, all task systems that are static-priority feasible (with respect to a given priority assignment) on a processor that has $(1 - \epsilon)$ times the computing capacity of the processor for which the task system is specified.

8.3 Future Research Agenda

In this section, we will briefly outline some ideas for future research.

8.3.1 Open Questions from Dissertation

Multiprocessor real-time scheduling theory is still a nascent area of research; therefore, there are many open areas of research not addressed in this dissertation. In this subsection, we briefly list some potential avenues for future research extending the results of this dissertation.

§ **Tightened conditions.** In Chapter 4, we show that our multiprocessor feasibility condition using demand-based load is at most a factor of approximately 1.61 from the optimal load condition. An interesting challenge is to decrease this gap in feasibility

analysis. Similarly, the question of whether similar gaps for schedulability analysis of various multiprocessor scheduling algorithms may be decreased is an open question.

§ **Partitioned scheduling of general task systems.** Despite our success at analyzing partitioned scheduling for the sporadic task model, we are currently unaware of non-trivial feasibility and schedulability algorithms for the partitioned scheduling of more general task models (such as GMF or DAG-based tasks). An open avenue of research is determining whether the techniques for partitioning sporadic task system apply to more general task models.

§ **Resource sharing in multiprocessor platforms.** A large body of research exists for resource sharing and synchronization in uniprocessor systems. Sophisticated protocols have been developed to ensure that the number of *priority inversions* are minimized (a priority inversion occurs when a lower-priority task "blocks" the execution of a higher-priority task due to synchronization). However, the issue of resource sharing in multiprocessor systems has not been addressed for the general task models discussed in this dissertation. Development of resource-sharing and synchronization protocols and analysis techniques are a prerequisite for the design of actual multiprocessor real-time systems.

§ **Multiprocessor scheduling under precedence constraints.** For many actual real-time systems, data must be communicated to a task τ_j before it may begin execution. The producer of this data is frequently another different task τ_i. In such a scenario, there is a *precedence constraint* between τ_i and τ_j. The existence of precedence constraints complicates schedulability analysis for a system. We would like to explore whether schedulability-analysis techniques developed in this dissertation could be meaningfully extended to account for precedence constraints between tasks.

8.3.2 Real-time Processor Virtualization with Resource Sharing

As mentioned in Section 8.2, virtualized real-time systems (i.e., open environments) have recently received much attention. There exists research that addresses the sharing of additional shared non-preemptable resources; however, most of the previously-proposed techniques place severe restrictions on the individual applications — in effect requiring that the resource-requesting jobs comprising these applications be made available in a first-come first-serve manner to the higher-level scheduler (as would happen, e.g., if the applications were scheduled using non-flexible, table-driven scheduling). We are currently developing an approach using sophisticated resource-sharing protocols to increase the schedulability of the entire open environment and decrease the complexity of system schedulability analysis. An interesting and challenging problem is to extend resource-sharing techniques for uniprocessor open environments to multiprocessor open environments. Also, the development of operating system support for developing applications in an open environment would pose many practical research problems.

8.3.3 Approximate Response-time Analysis for Uniprocessors

As mentioned in Section 8.2, some of our research has been on developing an FPTAS for schedulability analysis on uniprocessor systems. An interesting problem, related to schedulability analysis, is *response-time analysis*. Response-time analysis determines the maximum elapsed time from release of a task to its completion with respect to a given scheduling algorithm. The response-time analysis is useful in modeling the communication between tasks in a distributed real-time system. Typically, response-time

analysis must be performed multiple times to successfully obtain a useful communication model. Thus, a fast, tunable approximation of response-time analysis would increase the efficiency of such techniques. We have recently been exploring (Richard et al., 2007; Fisher et al., 2007b) whether an FPTAS for response-time analysis can be obtained for static-priority, uniprocessor systems, and there remains opportunities for further research in this topic.

8.4 Concluding Remarks

With the emergence of multicore and related technologies, the standard for future embedded real-time systems will be a multiprocessor platform. Due to the heterogeneity of system consumers, applications run upon these platforms are likely to be extremely diverse and characterized by complex software interactions (e.g., communication and resource-sharing). Current temporal analysis techniques cannot address many of these complex interactions on a multiprocessor system; this dissertation addresses some fundamental problems of analyzing complex multiprocessor real-time systems that were unanalyzable with previously known methods. Future research will continue to remove some of the simplifying assumptions from the real-time task models, thereby increasing the number of real-time systems that may utilize multiprocessor platforms.

Appendix A

Proof of Theorem 5.1

Section 5.1 introduced task system τ_{example} (given by Equation 5.1) used to prove that online multiprocessor scheduling of arbitrary and constrained task systems requires clairvoyance. In this appendix, we give a formal proof of Theorem 5.1. In other words, we prove the task system τ_{example} is feasible on two processors. In Section A.1, we informally outline our proof. In Section A.2, we introduce the formal notation involved in τ_{example}'s feasibility proof. In Section A.3, we give the entire feasibility proof.

A.1 Outline

The goal of Theorem 5.1 is to show that task system τ_{example} is feasible on two processors. However, we are unaware of any existing, non-trivial feasibility test for constrained-deadline task systems on a multiprocessor platform. Thus, we must tailor an argument specially for task system τ_{example}, and show that for every legal arrival sequence of jobs of τ_{example} there exists a schedule where no deadlines are missed. The following steps informally explain our proof of feasibility.

1. **Show that $\tau_{\text{example}} - \{\tau_6\}$ is feasible on two processors**: This can be shown by giving a partition of $\tau_{\text{example}} - \{\tau_6\}$ on two processors. The tasks individually assigned to a processor will be shown to be feasible on that processor with respect to the Deadline-Monotonic (DM) scheduling algorithm. Let the schedule

constructed by this approach be called S_I. Lemma A.2 corresponds to this step.

2. **Construct a modified schedule S'_I**: For any real-time instance $I \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_{\mathrm{example}})$, we construct a *global* schedule (i.e., non-partitioned) by moving as much work as possible to the first processor π_1. The construction for S'_I is given by Equation A.18. Lemma A.3 proves the validity of S'_I. In Lemmas A.4, A.5, A.6, A.7, and A.8, we prove several desirable properties that S'_I exhibits.

3. **Construct a schedule S''_I that leaves sufficient room for τ_6 to be completely assigned to the second processor**: We used the properties of S'_I (Lemmas A.4, A.5, A.6, A.7, and A.8) to show that a schedule S''_I can always be constructed leaves the second processor idle for four units between the release and deadline of a job of τ_6. Obviously, τ_6 can be completely assigned to these idle times. Therefore, τ_{example} is feasible on two unit-capacity processors (Theorem 5.1).

Please note that we only consider real-time instances in $\mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_{\mathrm{example}})$; the feasibility of any instance $I' \in \mathcal{I}^{\mathrm{S}}(\tau_{\mathrm{example}})$ follows from the fact that there exists an $I \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_{\mathrm{example}})$ 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).

In the next section, we discuss some additional notation used in this appendix. In Section A.3, we formally carry-out the steps outlined in this subsection.

A.2 Notation

In this section, we present notation for describing the behavior of a sporadic task system τ. The remainder of this appendix heavily relies on the notation presented in

Sections 1.3 and 1.4. In the remainder of this appendix, we will assume that τ is a constrained-deadline system (i.e., for all $\tau_i \in \tau$, $d_i \leq p_i$). For each $I \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau)$, let $I(\tau_i) \subseteq I$ denote the jobs generated by τ_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 A.1 (Job-Release Function) *If τ_i is in a scheduling window at time t in real-time instance I then $r_i(I,t)$ is the release time of the most recently released job of τ_i (with respect to time t). Otherwise, $r_i(I,t) = \infty$ if τ_i is not in a scheduling window at time t. More formally,*

$$r_i(I,t) \stackrel{def}{=} \begin{cases} A_k \ , & \text{if } \exists J_k \in I(\tau_i) \text{ such that } A_k \leq t \leq A_k + D_k \\ \infty \ , & \text{otherwise.} \end{cases} \tag{A.1}$$

Definition A.2 (Job-Deadline Function) *If τ_i is in a scheduling window at time t for real-time instance I then $d_i(I,t)$ is the absolute deadline of the most recently released job of τ_i (with respect to time t). Otherwise, $d_i(I,t) = -\infty$ if τ_i is not in a scheduling window at time t.*

$$d_i(I,t) \stackrel{def}{=} \begin{cases} A_k + D_k \ , & \text{if } \exists J_k \in I(\tau_i) \text{ such that } A_k \leq t \leq A_k + D_k \\ -\infty \ , & \text{otherwise.} \end{cases} \tag{A.2}$$

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

Definition A.3 (Active-Job Function) *If τ_i is in a scheduling window at time t for real-time instance I, the $\varphi_i(I,t)$ is current active job at time t. Otherwise,*

$\varphi_i(I,t) = \perp$, *if* τ_i *is not in a scheduling window at time* t.

$$\varphi_i(I,t) \stackrel{def}{=} \begin{cases} J_k \ , & \text{if } \exists J_k \in I(\tau_i) \text{ such that } A_k \leq t \leq A_k + D_k \\ \perp \ , & \text{otherwise.} \end{cases} \tag{A.3}$$

Similar to Definition 1.2 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 τ_i.

Definition A.4 (Task-Schedule Function) *$S_I(\pi_\ell, t, \tau_i)$ is an indicator function denoting whether task τ_i is scheduled at time t for schedule S_I. In other words,*

$$S_I(\pi_\ell, t, \tau_i) \stackrel{def}{=} \begin{cases} 1 \ , & \text{if } \exists J_k \in I(\tau_i) :: S_I(\pi_\ell, t, J_i) = 1 \\ 0 \ , & \text{otherwise.} \end{cases} \tag{A.4}$$

The next definition defines the work that task τ_i receives on π_ℓ over a given interval. The job work function (Definition 1.4) is used.

Definition A.5 (Task Work Function) *$W_i(S_I, \pi_\ell, t_1, t_2)$ denotes the amount of processor time that τ_i receives from schedule S_I on processor π_ℓ over the interval $[t_1, t_2)$ for real-time instance I. In other words,*

$$W_i(S_I, \pi_\ell, t_1, t_2) \stackrel{def}{=} \sum_{J_k \in I(\tau_i)} W(S_I, J_i, t_1, t_2). \tag{A.5}$$

The next function (Definition A.6) is useful for characterizing the maximum amount of processor time a task may receive over a given interval of time.

Definition A.6 (Execution-Bound Function) $\text{EBF}(\tau_i, t)$ *bounds the maximum amount of time that a* constrained *task τ_i (i.e., $d_i \leq p_i$) can execute over an interval of length t. Specifically, $\text{EBF}(\tau_i, t)$ is the maximum execution time of jobs of τ_i*

over the interval $[0, t]$ *over all real-time instances* I *and valid schedules. Formally,*

$$\text{EBF}(\tau_i, t) \stackrel{def}{=} \max \left\{ \begin{array}{l} \left\lfloor \frac{t}{p_i} \right\rfloor e_i + g_1\left(\left\lfloor \frac{t}{p_i} \right\rfloor, t\right) + g_2\left(\left\lfloor \frac{t}{p_i} \right\rfloor, t\right), \\ \\ \left(\left\lfloor \frac{t}{p_i} \right\rfloor - 1\right) e_i + g_1\left(\max\left(\left(\left\lfloor \frac{t}{p_i} \right\rfloor - 1\right), 0\right), t\right) \\ \qquad\qquad + g_2\left(\max\left(\left(\left\lfloor \frac{t}{p_i} \right\rfloor - 1\right), 0\right), t\right) \end{array} \right\}, \quad \text{(A.6)}$$

where

$$g_1(c, t) \stackrel{def}{=} \min\left(e_i, \max\left(0, t - c \times p_i - (p_i - d_i) - g_2(c, t)\right)\right) \quad \text{(A.7)}$$

and

$$g_2(c, t) \stackrel{def}{=} \min\left(e_i, t - c \times p_i\right). \quad \text{(A.8)}$$

The following claim provides an upper bound on the amount of time that τ_i may execute over any interval of length t, if c_i jobs are completely "contained" within the interval:

Claim A.1 *If* $I \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau)$ *is a real-time instance for constrained-deadline sporadic task system* τ*, and* $t_1, t_2 \in \mathbb{R}^+$ *such that* $0 \leq t_1 \leq t_2$*. Let* c_i *be the number of jobs of* τ_i *with arrival time in* $[t_1, t_2]$ *that arrive at least* p_i *time units before* t_2 *(i.e.,* $c_i \stackrel{def}{=} |\{J_k \in I(\tau_i) : t_1 \leq A_k < t_2 - p_i\}|$*.). Then, the following inequality quantifies the maximum amount of work that can occur over the interval* $[t_1, t_2)$ *on task* $\tau_i \in \tau$ *in any valid schedule* S_I *on platform* Π*:*

$$\sum_{\pi_\ell \in \Pi} W_i(S_I, \pi_\ell, t_1, t_2) \leq c_i \times e_i + g_1(c_i, t_2 - t_1) + g_2(c_i, t_2 - t_1). \quad \text{(A.9)}$$

Proof: Notice for $I(\tau_i)$ there is at most one job that arrives within p_i prior to t_1 (denote this job by J_{prior}), and at most one job that arrives with p_i prior to t_2 (denote this job by J_{after}). Let C_i denote the set $\{J_k \in I(\tau_i) : t_1 \leq A_k < t_2 - p_i\}$;

hence, $c_i = |C_i|$. The amount of execution by jobs of τ_i that can occur over the interval $[t_1, t_2)$ is:

$$\begin{array}{ll} \text{Work done by job } J_{\text{prior}} \text{ in the interval } [t_1, t_2) & + \\ \text{Execution requirement of jobs correspond to the arrivals in } C_i & + \\ \text{Work done by job } J_{\text{after}} \text{ in the interval } [t_1, t_2) & \end{array}$$

Obviously, the execution requirement of jobs of C_i is equal to $c_i \times e_i$; so, we will focus the maximum work contributed by the remaining two jobs. Assume that job J_{prior} is released at time $t_1 - (p_i - x_{\text{prior}})$ and job J_{after} is released at time $t_2 - x_{\text{after}}$ where $0 \leq x_{\text{prior}}, x_{\text{after}} < p_i$. If J_{after} (or J_{prior}) does not exist, we will use the convention that x_{after} (or x_{prior}, respectively) will be equal to zero which implies that the overhanging job does not execute during the interval $[t_1, t_2)$. Since there are c_i jobs arriving in-between jobs J_{prior} and J_{after}, the following inequality must hold by minimum inter-arrival separation parameter p_i (see Figure A.1 for a visual justification of this inequality):

$$\begin{array}{rrl} & (t_2 - x_{\text{after}}) - (t_1 - (p_i - x_{\text{prior}})) & \geq (c_i + 1)p_i \\ \Rightarrow & x_{\text{prior}} + x_{\text{after}} & \leq (t_2 - t_1) - c_i p_i. \end{array} \tag{A.10}$$

Since J_{prior} is released at time $t_1 - (p_i - x_{\text{prior}})$, its deadline occurs at $t_1 - (p_i - x_{\text{prior}}) + d_i$. Therefore, the most J_{prior} may execute during the interval $[t_1, t_2)$ is $\min(e_i, \max(0, x_{\text{prior}} - (p_i - d_i)))$. Since J_{after} is released at time $t_2 - x_{\text{after}}$, its deadline occurs at $t_2 - x_{\text{after}} + d_i$. The most J_{after} may execute during interval $[t_1, t_2)$ is $\min(e_i, x_{\text{after}})$. The maximum amount of time that jobs J_{prior} and J_{after} may execute in $[t_1, t_2)$ is

$$F(x_{\text{prior}}, x_{\text{after}}) \stackrel{\text{def}}{=} \min(e_i, \max(0, x_{\text{prior}} - (p_i - d_i))) + \min(e_i, x_{\text{after}}). \tag{A.11}$$

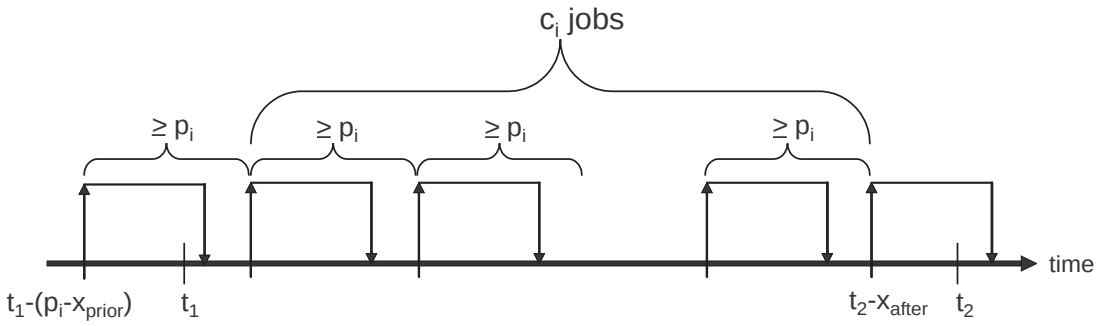


Figure A.1: The example release sequence shows that times $t_1 - (p_i - x_{\text{prior}})$ and $t_2 - x_{\text{after}}$ are separated by at least $(c_i + 1)p_i$ time. The "up-arrows" indicate a job release; the corresponding "down-arrows" indicate the job's absolute deadline. Inequality A.10 follows from the following facts: **(i)** c_i jobs of τ_i arrive between J_{prior} and J_{after} and each job arrival is separated by p_i time units; **(ii)** the successive job of τ_i after J_{prior} arrives at least p_i time units after J. (Note that if J_{prior} does not exist then by assumption x_{prior} is zero); and **(iii)** by assumption of the claim, the last job in the sequence of c_i jobs must arrive at least p_i time units before t_2 (this last fact is important if J_{after} does not exist and x_{after} is equal to zero).

The following values of x_{prior} and x_{after} maximize $F(x_{\text{prior}}, x_{\text{after}})$ (Equation A.11):

$$x^*_{\text{after}} = \min\left(e_i, (t_2 - t_1) - c_i p_i\right) \tag{A.12}$$

and

$$x^*_{\text{prior}} = (t_2 - t_1) - c_i p_i - x^*_{\text{after}}. \tag{A.13}$$

To see why the above values of x^*_{prior} and x^*_{after} maximize Equation A.11, let us consider a fixed x'_{after} set to a value different than x^*_{after}. We will show that for each fixed value of x'_{after} (and all possible values of x'_{prior} subject to Inequality A.10), $F(x^*_{\text{prior}}, x^*_{\text{after}}) \geq F(x'_{\text{prior}}, x'_{\text{after}})$. First, note that both $\min(e_i, \max(0, x_{\text{prior}} - (p_i - d_i)))$ and $\min(e_i, x_{\text{after}})$ are both non-decreasing, and increase at most linearly (with respect to x_{prior} and x_{after}, respectively).

In the first case, we consider values of x'_{after} exceeding x^*_{after}. Fix $x'_{\text{after}} = x^*_{\text{after}} + \gamma$ where $0 \leq \gamma \leq (t_2 - t_1) - c_i p_i - x^*_{\text{after}}$. The upper bound on γ follows from the fact

that the c_i jobs complete contained in $[t_1, t_2]$ require $c_i p_i$ time; the most we could increase x'_{after} to is bounded by the interval length $(t_2 - t_1)$ minus the execution of completely contained jobs $(c_i p_i)$ minus the value of x^*_{after}. If $(t_2 - t_1) - c_i p_i - x^*_{\text{after}} > 0$, then by the property of min function in Equation A.12, $x^*_{\text{after}} = e_i$; however, in this case, $e_i = \min(e_i, x^*_{\text{after}}) = \min(e_i, x^*_{\text{after}} + \gamma) = \min(e_i, x'_{\text{after}})$. So, $\min(e_i, x'_{\text{after}})$ does not exceed $\min(e_i, x^*_{\text{after}})$. Furthermore, since $\min(e_i, \max(0, x_{\text{prior}} - (p_i - d_i)))$ is non-decreasing, its maximum value is achieved when x'_{prior} is as large as possible, which is $x'_{\text{prior}} = (t_2 - t_1) - c_i p_i - x'_{\text{after}} = (t_2 - t_1) - c_i p_i - x^*_{\text{after}} - \gamma$. It is easy to see that $\min(e_i, \max(0, x^*_{\text{prior}} - (p_i - d_i))) \geq \min(e_i, \max(0, x'_{\text{prior}} - (p_i - d_i)))$; thus, $F(x^*_{\text{prior}}, x^*_{\text{after}}) \geq F(x'_{\text{prior}}, x'_{\text{after}})$ when x'_{prior} exceeds x^*_{prior}.

In the second case, consider values of x'_{after} at most x^*_{after}. Fix $x'_{\text{after}} = x^*_{\text{after}} - \gamma$ where $0 \leq \gamma \leq x^*_{\text{after}}$. Since $x^*_{\text{after}} \leq e_i$, then $\min(e_i, x'_{\text{after}}) = \min(e_i, x^*_{\text{after}} - \gamma) = \min(e_i, x^*_{\text{after}}) - \gamma$. However, $\min(e_i, \max(0, x_{\text{prior}} - (p_i - d_i)))$ only grows linearly with x_{prior}; so, $\min(e_i, \max(0, x'_{\text{prior}} - (p_i - d_i))) \leq \min(e_i, \max(0, x^*_{\text{prior}} - (p_i - d_i))) + \gamma$. Thus, $F(x'_{\text{prior}}, x'_{\text{after}}) \leq \min(e_i, x^*_{\text{after}}) + \min(e_i, \max(0, x^*_{\text{prior}} - (p_i - d_i))) = F(x^*_{\text{prior}}, x^*_{\text{after}})$. Therefore, x^*_{prior} and x^*_{after} maximize Equation A.11.

Observe that when x_{prior} and x_{after} are set according to Equations A.13 and A.12, Equation A.11 is equal to $g_1(c_i, t_2 - t_1) + g_2(c_i, t_2 - t_1)$ which immediately implies the claim. ■

Claim A.1 provides an upper bound on the amount of work with respect to c_i, the number of jobs completely contained in an interval $[t_1, t_2)$. We will now show, in the following lemma, that EBF is an upper bound on the amount of work over all possible values of c_i.

Lemma A.1 *Let $S_I \in \mathbb{S}_{I,\Pi}$ be a schedule for constrained-deadline task system τ on platform Π (with respect to release sequence $I \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau)$) that satisfies Conditions 1*

and 2 of validity (Definition 1.6). Then, for all $t_1, t_2 \in \mathbb{R}^+$ such that $0 \leq t_1 \leq t_2$,

$$\text{EBF}(\tau_i, t_2 - t_1) \geq \sum_{\pi_\ell \in \Pi} W_i(S_I, \pi_\ell, t_1, t_2). \tag{A.14}$$

Proof: Observe that $c_i \leq \left\lfloor \frac{t_2 - t_1}{p_i} \right\rfloor$. We will now provide a case analysis based on c_i of the maximum work over interval $[t_1, t_2)$. There are three cases we will consider:

1. $c_i \leq \left\lfloor \frac{t_2 - t_1}{p_i} \right\rfloor - 2$,
2. $c_i = \left\lfloor \frac{t_2 - t_1}{p_i} \right\rfloor - 1$, and
3. $c_i = \left\lfloor \frac{t_2 - t_1}{p_i} \right\rfloor$.

We prove in each of the above cases that Equation A.14 holds. To see that Equation A.14 holds for Case 1, observe that $W_i(S_I, \pi_\ell, t_1, t_2) \leq \left\lfloor \frac{t_2 - t_1}{p_i} \right\rfloor \times e_i$; this fact follows because the "overhanging" jobs of interval $[t_1, t_2)$ contribute at most $2e_i$ units of work while the "completely contained" jobs of $[t_1, t_2)$ contribute $(\left\lfloor \frac{t_2 - t_1}{p_i} \right\rfloor - 2) \times e_i$. From Equation A.6 (Definition A.6), it follows that $\text{EBF}(\tau_i, t_2 - t_1) \geq \left\lfloor \frac{t_2 - t_1}{p_i} \right\rfloor \times e_i$ because functions g_1 and g_2 will both be non-negative; so, Equation A.14 holds for Case 1.

By Claim A.1, $\text{EBF}(\tau_i, t_2 - t_1)$ is an upper bound on $W_i(S_I, \pi_\ell, t_1, t_2)$ in both Cases 2 and 3; therefore, Equation A.14 holds for all possible cases and the lemma follows. ■

A.3 Proof

In this section, we prove Theorem 5.1 by following the steps outlined in Section A.1. Section A.3.1 gives the construction for schedule S_I. Section A.3.2 describes the construction of schedule S_I' and gives several lemmas describing important properties

of S'_I. Section A.3.3 contains the proof of Theorem 5.1 by showing that a schedule S''_I can be constructed to accommodate τ_6 on processor π_2.

A.3.1 Construction of Schedule S

The first step of the outline (Section A.1) of the proof requires us to show that $\tau_{\text{example}} - \{\tau_6\}$ is feasible on two processors. We can easily obtain feasibility of this task system by partitioning $\tau_{\text{example}} - \{\tau_6\}$ into two sets and scheduling each subset on its own processor using a uniprocessor scheduling algorithm. For simplicity of analysis, we will use DM on each processor.

Audsley *et al.* (Audsley et al., 1991) developed a test to determine whether each task in a *constrained-deadline* task system can always meet all deadlines. Let $\mathbf{T}_{H_i}$ be the set of tasks with priority greater than or equal to task τ_i under the DM priority assignment. The following theorem restates their result.

Theorem A.1 (from (Audsley et al., 1991)) *In a constrained-deadline, sporadic task system, task τ_i always meets all deadlines using* DM *on a preemptive uniprocessor if and only if $\exists t \in (0, d_i]$ such that*

$$\text{CBF}(\tau_i, t) \stackrel{def}{=} \sum_{\tau_j \in \mathbf{T}_{H_i}} \text{RBF}(\tau_j, t) + e_i \leq t. \tag{A.15}$$

■

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

Lemma A.2 *$\tau_{\text{example}} - \{\tau_6\}$ is feasible on a multiprocessor platform composed of two unit-capacity processors.*

Proof: Define the following partition of $\tau_{\text{example}} - \{\tau_6\}$:

$$\tau_A \stackrel{\text{def}}{=} \{\tau_1, \tau_4\}, \tag{A.16}$$

and

$$\tau_B \stackrel{\text{def}}{=} \{\tau_2, \tau_3, \tau_5\}. \tag{A.17}$$

Assign τ_A to π_1 and τ_B to π_2. It is easy to verify (by Theorem A.1) that τ_A and τ_B are feasible with respect to their assigned processors.

■

Let S_I be the schedule constructed by partitioned DM for task system $\tau_{\text{example}} - \{\tau_6\}$ with partitions τ_A and τ_B. From the previous argument, S_I is valid for any $I \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau_{\text{example}} - \{\tau_6\})$.

A.3.2 Construction of Schedule S'_I

We now 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 π_2 as possible. To accomplish this goal, for every idle instant on processor π_1 in schedule S_I, we move a task in its scheduling window on π_2 to π_1 (if such a task exists). The construction "builds" schedule S'_I for processor π_1, first. After $S'_I(\pi_1, t)$ is constructed, then S'_I is constructed for π_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).

In schedule $S'_I(\pi_1, t)$, tasks of set τ_A (tasks τ_1 and τ_4) execute at exactly the same times as they did in schedule $S_I(\pi_1, t)$. However, the tasks of set τ_B move as much execution as possible (without disturbing tasks of τ_A) from processor π_2 to processor π_1. Consider an arbitrary time t; $S'_I(\pi_1, t)$ is constructed using the following rules:

1. If at time t processor π_1 is busy executing a job from tasks of τ_A in schedule S_I, $S'_I(\pi_1, t)$ equals $S_I(\pi_1, t)$.

2. If processor π_1 is idle at time t in schedule S_I, then:

   (a) If task τ_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 π_1, then S'_I at time t is set to the current active job of τ_5 – i.e $S'_I(\pi_1, t) = \varphi_5(I, t)$;

   (b) *else*, if task τ_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 π_1, then S'_I at time t is set to the current active job of τ_3 – i.e $S'_I(\pi_1, t) = \varphi_3(I, t)$;

   (c) *else*, if task τ_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 π_1, then S'_I at time t is set to the current active job of τ_2 – i.e $S'_I(\pi_1, t) = \varphi_2(I, t)$;

   (d) *else*, leave processor π_1 idle.

The execution of jobs of tasks in τ_A that could not be moved to processor π_1 is executed on processor π_2 (with the added constraint that a task does not execute in parallel with itself). Figure A.2 presents a visual example comparing schedules S_I and S'_I for a possible release sequence. The following construction is the inductive formal definition of the modified schedule for all $I \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_{\mathrm{example}} - \{\tau_6\})$ 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 π_1.

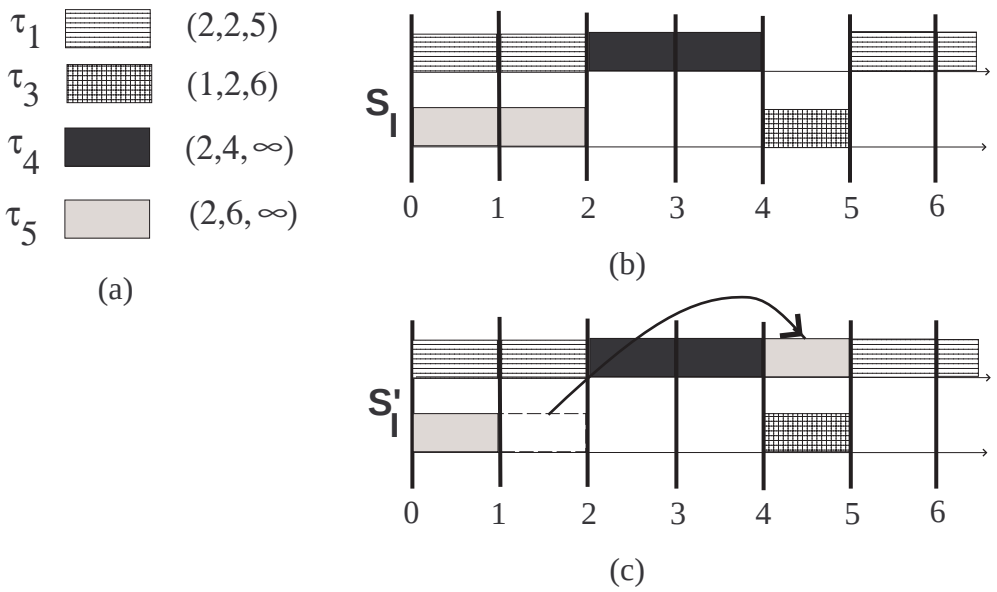


Figure A.2: Consider a release sequence containing tasks τ_1, τ_3, τ_4, and τ_5. **(a)** replicates the legend for these tasks. Let τ_1, τ_4, and τ_5 release jobs at time $t = 0$; τ_3 releases a job at $t = 4$; and, τ_1 releases a second job at $t = 5$. **(b)** presents schedule S_I. **(c)** presents schedule S'_I. Note that the execution of τ_5 in the interval $[1, 2)$ is moved from the second processor to $[4, 5)$ on the first processor.

$$S'_I(\pi_1, t) \stackrel{\text{def}}{=} \begin{cases} S_I(\pi_1, t)\ , & \text{if } S_I(\pi_1, t) \neq \perp, \\ \varphi_5(I, t)\ , & \text{if } r_5(I, t) < \infty \text{ and} \\ & W_5(S'_I, \pi_1, r_5(I, t), t) < e_5, \\ \varphi_3(I, t)\ , & \text{if } r_3(I, t) < \infty \text{ and} \\ & W_3(S'_I, \pi_1, r_3(I, t), t) < e_3, \\ \varphi_2(I, t)\ , & \text{if } r_2(I, t) < \infty \text{ and} \\ & W_2(S'_I, \pi_1, r_2(I, t), t) < e_2, \\ \perp\ , & \text{otherwise} \end{cases} \tag{A.18}$$

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

Lemma A.3 *S'_I is valid for any $I \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_{\mathrm{example}} - \{\tau_6\})$.*

Proof: The first condition of validity (Definition 1.6) for schedule S'_I is easily seen to be satisfied by noting that a task from $\tau_A = \{\tau_1, \tau_4\}$ is scheduled according to S_I which is valid. Validity Condition 1 is satisfied for tasks in $\tau_B = \{\tau_2, \tau_3, \tau_5\}$ since S'_I only schedules a task $\tau_i \in \tau_B$ on π_1 if it is in a scheduling window$(r_i(I,t) < d_i(I,t))$. A task $\tau_i \in \tau_B$ is only scheduled on π_2 in S'_I if it was already scheduled on π_2 in S_I (which is valid).

The second condition of validity follows vacuously from the definition of S'_I. To show the third condition we must show for any $J_k \in I(\tau_A) \cup I(\tau_B)$ that was generated by $\tau_i \in \tau_A \cup \tau_B$ satisfies $e_i \leq \sum_{\pi_\ell \in \Pi} W_i(S'_I, \pi_\ell, A_k, A_k + D_k) \leq e_i$. First, note that for any $J_k \in I(\tau_A)$, the second condition of validity follows immediately from the validity of S_I because all jobs in $I(\tau_A)$ are scheduled exactly the same in S'_I as S_I. So, assume that $J_k \in I(\tau_B)$. Also, for J_k it is the case that E_k equals e_i, since $I \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau)$.

Let us first show that $e_i \leq \sum_{\pi_\ell \in \Pi} W_i(S'_I, \pi_\ell, A_k, A_k + D_k)$ for each $J_k \in I(\tau_B)$. Assume that there exists $J_k \in I(\tau_B)$ such that $e_i > \sum_{\pi_\ell \in \Pi} W_i(S'_I, \pi_\ell, A_k, A_k + D_k)$. Assume that J_k was generated by $\tau_i \in \tau_B$. Observe there must be a *minimum* time instant t' in the interval $[A_k, A_k + D_k]$ such that $e_i > \sum_{\pi_\ell \in \Pi} W_i(S'_I, \pi_\ell, A_k, t')$ and either $S'_I(\pi_1, t', \tau_i) = 1$ or $S'_I(\pi_2, t', \tau_i) = 1$. Regardless of which processor τ_i executes on at time t', there are two cases with respect to the amount of work that has been done on processor π_2 up until time t'. Either τ_i has executed on processor π_2 (Case 1), or it has not (Case 2). We will show in both cases a contradiction arises.

1. $W_i(S'_I, \pi_2, A_k, t') > 0$: In this case, let $t'' \stackrel{\mathrm{def}}{=} \max\{t \in [A_k, t'] : S'_I(\pi_2, t, \tau_i) = 1\}$. Note that t'' must exist and $S'_I(\pi_2, t'', \tau_i) = 1$ as $W_i(S'_I, \pi_2, A_k, t') > 0$. However, $e_i > \sum_{\pi_\ell \in \Pi} W_i(S'_I, \pi_\ell, A_k, t')$ and $A_k = r_i(I,t)$ implies that

$W_i(S'_I, \pi_1, r_i(I,t'), d_i(I,t')) + W_i(S'_I, \pi_2, r_i(I,t'), t'') > e_i$. This contradicts Equation A.18 and $S'_I(\pi_2, t'', \tau_i) = 1$.

2. $W_i(S'_I, \pi_2, A_k, t') = 0$: This implies that $W_i(S'_I, \pi_1, A_k, t') > e_i$. If $S'_I(\pi_1, t', \tau_i) = 1$, this contradicts Equation A.18. If $S'_I(\pi_2, t', \tau_i) = 1$, $W_i(S'_I, \pi_1, A_k, t') > e_i$ implies $W_i(S'_I, \pi_1, r_i(I,t'), d_i(I,t')) + W_i(S'_I, \pi_2, r_i(I,t'), t') > e_i$ which again contradicts Equation A.18.

Since in both cases a contradiction arises, such a t' cannot exist and $e_i \leq \sum_{\pi_\ell \in \Pi} W_i(S'_i, \pi_\ell, A_k, A_k + D_k)$.

Let us show that $e_i \geq \sum_{\pi_\ell \in \Pi} W_i(S'_I, \pi_\ell, A_k, A_k + D_k)$. Assume that there exists $J_k \in I(\tau_B)$ such that $e_i < \sum_{\pi_\ell \in \Pi} W_i(S'_I, \pi_\ell, A_k, A_k + D_k)$. So,

$$\forall t \in [A_k, A_k + D_k] : \; W_i(S'_I, \pi_1, r_i(I,t), d_i(I,t)) + W_i(S'_I, \pi_2, r_i(I,t), t) < e_i. \quad \text{(A.19)}$$

(Again, note that $r_i(I,t)$ equals A_k and $d_i(I,t)$ equals $A_k + D_k$). Let $P_i \stackrel{\text{def}}{=} \{t \in [A_k, A_k + D_k] : S_I(\pi_2, t, \tau_i) = 1\}$. For all $t \in P_i$, $S'_I(\pi_2, t, \tau_i) = 0$ if and only if $S'(\pi_1, t, \tau_i) = 1$ (by Equations A.18 and A.19). This implies that $\sum_{\pi_\ell \in \Pi} W_i(S'_I, \pi_\ell, A_k, A_k + D_k) = \sum_{\pi_\ell \in \Pi} W_i(S_I, \pi_\ell, A_k, A_k + D_k)$. By the validity of S_I, $\sum_{\pi_\ell \in \Pi} W_i(S'_I, \pi_\ell, A_k, A_k + D_k) = e_i$. This contradicts our assumption; therefore, $e_i \geq \sum_{\pi_\ell \in \Pi} W_i(S'_I, \pi_\ell, A_k, A_k + D_k)$. Condition 3 of validity and the lemma follows. ■

We now wish to prove several lemmas which characterize the properties of schedule S'_I with respect to the interval in which τ_6 is in a scheduling window. The main observation from these properties is that S'_I can be modified to provide enough idle instants on processor π_2 to successfully schedule τ_6.

We will first quantify the maximum amount of τ_5's computation that has been moved to processor π_1 in the schedule S'_I. Let t_5 be the arrival time A_k of some job $J_k \in I(\tau_5)$. The following lemma bounds the amount of processor π_1's computation

that the set of tasks τ_A can consume over the interval $[t_5, t_5+6]$ (i.e., the time interval during which τ_5 is in a scheduling window).

Lemma A.4 *For all* $I \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_{\mathrm{example}})$, $\sum_{\tau_i \in \tau_A} W_i(S'_I, \pi_1, t_5, t_5+6) \leq 5$.

Proof: By the synchronous arrival sequence:

$$W_1(S'_I, \pi_1, t_5, t_5+6) \leq \mathrm{EBF}(\tau_1, 6) = 3$$

and,

$$W_4(S'_I, \pi_1, t_5, t_5+6) \leq 2 \ \ (\text{since } p_4 = 100).$$

Thus,$\sum_{\tau_i \in \tau_A} W_i(S'_I, \pi_1, t_5, t_5+6) \leq 5$. ■

The next lemma shows that the work done by task τ_5 on processor π_2 over the interval $[t_5, t_5+6]$ in schedule S'_I does not exceed one.

Lemma A.5 *For all* $I \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}(\tau_{\mathrm{example}})$, $W_5(S'_I, \pi_2, t_5, t_5+6) \leq 1$.

Proof: By Lemma A.4, $\sum_{\tau_i \in \tau_A} W_i(S'_I, \pi_1, t_5, t_5+6) \leq 5$. This implies that the set of idle points $t \in [t_5, t_5+6]$ where $S_I(\pi_1, t) = \perp$ has cardinality at least one. By Equation A.18, τ_5 will execute whenever τ_1 and τ_4 are not executing on π_1; thus, $W_5(S'_I, \pi_1, t_5, t_5+6) \geq 1$. By validity of S'_I, $W_5(S'_I, \pi_2, t_5, t_5+6) \leq 1$. ■

The goal of our argument is to show that there exists a multiprocessor schedule for $\tau_{\mathrm{example}} - \{\tau_6\}$ which leaves processor π_2 idle for at least four time units over the interval during which τ_6 is in a scheduling window. Our argument relies on either having sufficient idle time in schedule S'_I to execute τ_6 completely on π_2, or being able to move work of tasks τ_2, τ_3, and τ_5 in schedule S'_I to accommodate task τ_6 (by creating a new schedule S''_I). Therefore, it would be useful to reason about intervals during which π_1 is continuously busy executing only tasks of $\tau_A \cup \{\tau_5\}$.

Observe that $\text{EBF}(\tau_1, 8) + \text{EBF}(\tau_4, 8) + \text{EBF}(\tau_5, 8) = 4 + 2 + 2 = 8$. However, task τ_5 may not always be able to entirely execute on processor π_1 in schedule S'_I. For any t_5 equal to the arrival time of some job in $I(\tau_5)$, let $\alpha(t_5, S'_I)$ be the amount of time that τ_5 executes on processor π_2 in schedule S'_I for real-time instance I. That is,

$$\alpha(t_5, S'_I) \stackrel{\text{def}}{=} W_5(S'_I, \pi_2, t_5, t_5 + 6). \tag{A.20}$$

Therefore, the amount time that τ_5 can execute on processor π_1 over any interval for instance I is at most $2 - \alpha(t_5, S'_I)$.

The next lemma will describe the implications of processor π_1 being continuously busy during the interval $[t_5, t_5 + 6]$ where τ_5 must partially execute on processor π_2. The implications of a continuously busy processor π_1 for $[t_5, t_5 + 6]$ is that τ_4 must execute during this interval. Let t_4 be an arrival time of some job of τ_4 in I where $[t_5, t_5 + 6] \cap [t_4, t_4 + 4] \neq \emptyset$. The next lemma will show that π_1 is continuously busy in the interval $[t_4, t_4 + 4]$ as well.

Lemma A.6 *For all* $I \in \mathcal{I}^{\text{S}}_{\text{WCET}}(\tau_{\text{example}} - \{\tau_6\})$, *if* $\sum_{\tau_j \in \tau_A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_5, t_5+6) = 6$ *and* $\alpha(t_5, S'_I) > 0$, *then*

$$\exists t_4 :: ([t_4, t_4 + 4] \cap [t_5, t_5 + 6] \neq \emptyset) \wedge \left(\sum_{\tau_j \in \tau_A \cap \{\tau_5\}} W_j(S'_I, \pi_1, t_4, t_4 + 4) = 4 \right). \tag{A.21}$$

Proof: Observe that $W_4(S'_I, \pi_1, t_5, t_5+6) \leq 2$ and $W_5(S'_I, \pi_1, t_5, t_5+6) \leq 2-\alpha(t_5, S'_I)$, by $e_4 = 2$, $e_5 = 2$, and definition of α. This implies

$$W_1(S'_I, \pi_1, t_5, t_5 + 6) \geq 6 - 2 - (2 - \alpha(t_5, S'_I)) = 2 + \alpha(t_5, S'_I) > 2. \tag{A.22}$$

Because $e_1 = 2$, Equation A.22 means that there exist two jobs J_1 and J_2 of

τ_1 that have scheduling windows over the interval $[t_5, t_5 + 6]$. Let r_1^1 and r_2^1 be the release times of J_1 and J_2 in instance $I(\tau_1)$, respectively. Observe that since $\frac{e_1}{d_1} = 1$, J_1 continuously executes on processor π_1 over the interval $[r_1^1, r_1^1 + 2]$, and J_2 executes continuously over the interval $[r_2^1, r_2^1 + 2]$. Also, $t_5 \leq r_1^1 + 2 \leq r_2^1 \leq t_5 + 6$. Because $p_1 = 5$, $|r_2^1 - (r_1^1 + 2)| \geq 3$ and $W_1(S'_I, \pi_1, r_1^1 + 2, r_2^1) = 0$. Since $[r_1^1 + 2, r_2^1] \subset [t_5, t_5 + 6]$ and $\alpha(t_5, S'_I) > 0$, the most τ_5 can execute on processor π_1 in the interval $[r_1^1 + 2, r_2^1]$ is $2 - \alpha(t_5, S'_I)$. Therefore, $W_4(S'_I, \pi_1, r_1^1 + 2, r_2^1) > 1$ which implies there exists a arrival time t_4 of some job of $I(\tau_4)$ such that $[t_4, t_4 + 4] \cap [t_5, t_5 + 6] \neq \emptyset$.

From this it follows that there are two cases. We will show that the cases imply Equation A.21. The cases are:

1. $[t_4, t_4 + 4] \subset [t_5, t_5 + 6]$: By the antecedent of the lemma, π_1 is continuously busy executing jobs of $I(\tau_A \cap \{\tau_5\})$. Thus, Equation A.21 follows trivially.

2. $|[t_4, t_4 + 4] \cap [t_5, t_5 + 6]| < 4$: Given this case, there are two possibilities. Either the job of τ_4 is released before t_5 or it is released after t_5 (otherwise, $[t_4, t_4 + 4] \subset [t_5, t_5 + 6]$). More formally, the subcases are:

   a) $t_4 < t_5 < t_4 + 4$: In this case, $[t_4, t_4 + 4] \cap [r_1^1, r_1^1 + 2] \neq \emptyset$, because $W_4(S'_I, \pi_1, , r_1^1 + 2, r_2^1) > 1$ and $[r_1^1, r_1^1 + 2] \cap [t_5, t_5 + 6] \neq \emptyset$. This implies three additional subcases:

      i) τ_4 *executes entirely after* $r_1^1 + 2$: This implies that $t_4 \in [r_1^1, r_1^1 + 2]$; otherwise, there would not be enough execution left in $[r_1^1, t_4 + 4]$ for τ_4 to execute. Since $t_4 < t_5$ in this case, $r_1^1 < t_4 < t_5$. Thus, the interval $[t_4, t_4 + 4]$ is a subset of $[r_1^1, t_5 + 6]$. Since π_1 is continuously busy executing J_1 during $[r_1^1, r_1^1 + 2]$ and by the antecedent of the lemma π_1 is continuously busy executing jobs of $\tau_A \cap \{\tau_5\}$ during $[t_5, t_5 + 6]$, it must be that π_1 is also continuously busy executing jobs of $\tau_A \cap \{\tau_5\}$

in the interval $[t_4, t_4 + 4]$. This implies Equation A.21.

ii) τ_4 *executes both before* r_1^1 *and after* r_1^1+2: Observe that job J_1 executes continuously over $[r_1^1, r_1^1 + 2]$. Since τ_4 executes both before and after r_1^1 and S'_I is valid, it follows that $[r_1^1, r_1^1 + 2] \subset [t_4, t_4 + 4]$. Thus, τ_4 is continuously executed on processor π_1 over the intervals $[t_4, r_1^1]$ and $[r_1^1 + 2, t_4 + 4]$. Thus,

$$\begin{aligned}
&W_4(S'_I, \pi_1, t_4, r_1^1) + W_1(S'_I, \pi_1, r_1^1, r_1^1 + 2) + \\
&\qquad W_4(S'_I, \pi_1, r_1^1 + 2, t_4 + 4) \\
&= (r_1^1 - t_4) + e_1 + ((t_4 + 4) - (r_1^1 + 2)) \\
&= e_1 + e_4 \\
&= 2 + 2 = 4
\end{aligned}$$

This implies Equation A.21.

iii) τ_4 *executes entirely before* r_1^1: This case is impossible because $W_4(S'_I, \pi_1, r_1^1 + 2, r_2^1) > 1$.

b) $t_5 + 2 < t_4 < t_5 + 6$: Symmetric to Case a.

■

Observe that the longest possible interval in which processor π_1 is continuously busy executing the jobs of τ_1, τ_4, and τ_5 is of length $8 - \alpha(t_5, S'_I)$ (due to the fact that both τ_4 and τ_5 can execute at most one job in any 100 time unit interval). The longest possible continuously busy interval of jobs of these tasks on processor π_1 contains two jobs of τ_1, one job of τ_4, and $2 - \alpha(t_5, S'_I)$ units of a job of τ_5. We will now use Lemma A.6 to show that if all intervals of length $8 - \alpha(t_5, S'_I)$ that contain $[t_5, t_5 + 6]$ are not continuously busy on processor π_1, then we can fit τ_5 entirely on π_1 (i.e., $\alpha(t_5, S'_I) = 0$). The next lemma formally states this observation.

Lemma A.7 *If for all* $t' \in [t_5 - 2 + \alpha(t_5, S'_I), t_5]$

$$\sum_{\tau_j \in \tau_A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t', t' + 8 - \alpha(t_5, S'_I)) < 8 - \alpha(t_5, S'_I),$$

then $\alpha(t_5, S'_I) = 0$.

Proof: The proof is by contradiction. Assume that the antecedent is true for a given I, but $\alpha(t_5, S'_I) > 0$. Since we cannot move all of τ_5's execution in the interval $[t_5, t_5 + 6]$ to processor π_1, this implies that

$$\sum_{\tau_j \in \tau_A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_5, t_5 + 6) = 6. \tag{A.23}$$

Observe that the antecedent of Lemma A.6 is thus satisfied. Therefore, by similar reasoning, two different jobs of τ_1 must have scheduling windows in the interval $[t_5, t_5 + 6]$. J_1 and J_2 execute continuously in intervals $[r_1^1, r_1^1 + 2]$ and $[r_2^1, r_2^1 + 2]$ (borrowing notation from Lemma A.6). From Equation A.23, τ_1, τ_4, and τ_5 execute continuously in the interval $[\min(r_1^1, t_5), \max(r_2^1 + 2, t_5 + 6)]$. By Lemma A.6, there exists t_4 that corresponds to an arrival job of τ_4 with $[t_4, t_4 + 4] \cap [t_5, t_5 + 6] \neq \emptyset$. Furthermore, π_1 is continuously busy executing jobs of $\tau_A \cup \{\tau_5\}$ in the interval $[t_4, t_4 + 4]$. Since each of these intervals overlap and are continuously busy, the following interval is continuously busy,

$$[\min(r_1^1, t_5, t_4), \max(r_2^1 + 2, t_5 + 6, t_4 + 4)].$$

This interval obviously includes two jobs of τ_1, one job of τ_4, and $2 - \alpha(t_5, S'_I)$ units of a job of τ_5. Define,

$$t_{start} \stackrel{\text{def}}{=} \min(r_1^1, t_5, t_4), \tag{A.24}$$

and

$$t_{end} \stackrel{\text{def}}{=} \max(r_2^1 + 2, t_5 + 6, t_4 + 4). \tag{A.25}$$

Then,

$$\sum_{\tau_j \in \tau_A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_{start}, t_{end}) = 8 - \alpha(t_5, S'_I)$$

However, observe that $t_{start} \in [t_5 - 2 + \alpha(t_5, S'_I), t_5]$; otherwise, processor π_1 would not be continuously busy over the interval $[t_5, t_5+6]$, contradicting Equation A.23. We have thus shown that a continuously busy interval on processor π_1 of size $8-\alpha(t_5, S'_I)$ exists if $\alpha(t_5, S'_I) > 0$. This contradicts our assumption; therefore, $\alpha(t_5, S'_I) = 0$. ■

In order to achieve our stated goal (i.e., processor π_2 has enough idle instances to successfully schedule τ_6), we must show that there is still enough idle instances on processor π_1 to move the execution of τ_B. In the next lemma, we show that any time interval in which π_1 is continuously busy executing jobs of $\tau_A \cup \{\tau_5\}$ must be surrounded by continuously idle intervals (with respect to jobs of $\tau_A \cup \{\tau_5\}$). These idle intervals must be of length at least two. The next lemma formally states this observation.

Lemma A.8 *If* $\alpha(t_5, S'_I) > 0$ *then*

$$\sum_{\tau_j \in \tau_A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_{start} - 2, t_{start}) = 0, \tag{A.26}$$

and

$$\sum_{\tau_j \in \tau_A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_{start} + (8 - \alpha(t_5, S'_I)), t_{start} + (8 - \alpha(t_5, S'_I) + 2) = 0. \tag{A.27}$$

Proof: We will prove that $\alpha(t_5, S'_I) > 0$ implies Equation A.26. Equation A.27 can

be proven symmetrically. Observe that

$$W_1(S'_I, \pi_1, r_1^1 - 3, r_1^1) = 0 \tag{A.28}$$

because $p_1 = 5$ and $d_1 = 2$. From Lemma A.7, $\alpha(t_5, S'_I) > 0$ implies that

$$\begin{aligned} & \textstyle\sum_{\tau_j \in \tau_A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_{start}, t_{end}) = 8 - \alpha(t_5, S'_I) \\ \Rightarrow \quad & \textstyle\sum_{\tau_j \in \{\tau_4, \tau_5\}} W_j(S'_I, \pi_1, r_1^1 + 2, r_2^1) \geq 3. \end{aligned}$$

The last implication follows from reasoning in Lemma A.6. Since at least three units of τ_4 and τ_5 must execute in the interval $[r_1^1 + 2, r_2^1]$, this leaves at most $1 - \alpha(t_5, S'_I)$ units left to execute either before r_1^1 and/or after $r_2^1 + 2$. This implies

$$t_{start} \geq r_1^1 - 1 + \alpha(t_5, S'_I). \tag{A.29}$$

Equations A.28 and A.29 imply that the latest another job of τ_1 could execute prior to t_{start} is $t_{start} - 2$. Since τ_5 and τ_4 have periods equal to 100, and they release jobs contained within $[t_{start}, t_{end}]$, they are not active in the interval $[t_{start} - 2, t_{start}]$. Therefore,

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

■

A.3.3 Construction of Schedule S''_I and Proof of Theorem 5.1

We now have sufficient tools to successfully prove Theorem 5.1. We will show that if S'_I does not have enough idle instants to schedule τ_6 entirely on processor π_2, then we can create a schedule S''_I.

Proof of Theorem 5.1

Let t_6 be the arrival of any job of task τ_6 in real-time instance $I \in \mathcal{I}^{\mathrm{S}}_{\mathrm{WCET}}$. Consider S'_I constructed in Section A.3.2 (Equation A.18). If $\sum_{\tau_j \in \tau_{\text{example}} - \{\tau_6\}} W_j(S'_I, \pi_2, t_6, t_6 + 8) \leq 4$, then we are done (i.e., there is sufficient idle time on π_2 in S'_I to execute the job of τ_6 for 4 time units in $[t_6, t_6 + 8]$). So assume that

$$\sum_{\tau_j \in \tau_{\text{example}} - \{\tau_6\}} W_j(S'_I, \pi_2, t_6, t_6 + 8) > 4. \tag{A.30}$$

Observe that

$$W_2(S'_I, \pi_2, t_6, t_6 + 8) \leq \textsc{ebf}(\tau_2, 8) = 2, \tag{A.31}$$

and

$$W_3(S'_I, \pi_2, t_6, t_6 + 8) \leq \textsc{ebf}(\tau_3, 8) = 2, \tag{A.32}$$

Equations A.30, A.31, and A.32 together imply that $[t_6, t_6 + 8]$ must overlap with the scheduling window of a job J_k of τ_5. Let t_5 be the arrival time of J_k. Furthermore, J_k must execute some on processor π_2, so $\alpha(t_5, S'_I) > 0$. From Equation A.30, $[t_5, t_5 + 6] \cap [t_6, t_6 + 8] \neq \emptyset$. Since $\alpha(t_5, S'_I) > 0$, Lemma A.7 implies that $[t_5, t_5 + 6]$ overlaps with two jobs of τ_1 and one job of τ_4; Using the definition of t_{start} and t_{end} from Equations A.24 and A.25, $\sum_{\tau_j \in \tau_A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_{start}, t_{end}) = 8 - \alpha(t_5, S'_I)$.

Considering the intervals $X_{start} \stackrel{\text{def}}{=} [t_{start}, t_{start} + 8 - \alpha(t_5, S'_I)]$, $X_5 \stackrel{\text{def}}{=} [t_5, t_5 + 6]$, and $X_6 \stackrel{\text{def}}{=} [t_6, t_6 + 8]$ (recall that $X_5 \subset X_{start}$ and $X_6 \cap X_5 \neq \emptyset$), we provide three comprehensive cases for the overlap of these intervals. In each of these cases (and their subcases), we show that the subcase is impossible given that τ_6 does not fit (Equation A.30), or it is possible to construct a schedule S''_I such that $\sum_{\tau_j \in \tau_{\text{example}} - \{\tau_6\}} W_j(S''_I, \pi_2, t_6, t_6 + 8) \leq 4$. The cases for overlap are as follows.

1. $t_6 \leq t_{start} \leq t_6 + \alpha(t_5, S'_I)$.

2. $t_{start} < t_6$:

   a) $t_5 > t_6 - 2$.

   b) $t_6 - 6 < t_5 \leq t_6 - 2$.

3. $t_{start} > t_6 + \alpha(t_5, S'_I)$:

   a) $t_5 < t_6 + 4$.

   b) $t_6 + 8 > t_5 \geq t_6 + 4$.

We invite the reader to verify that this list of cases is comprehensive. Proof that each of these cases are either impossible (Case 1) or can be modified to create a valid schedule S'' (Cases 2 and 3) follows.

1. $t_6 \leq t_{start} \leq t_6 + \alpha(t_5, S'_I)$: This implies that $X_{start} \subset X_6$. By Lemma A.8,

$$\sum_{\tau_j \in \tau_A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_6, t_{start}) = 0, \tag{A.33}$$

   and

$$\sum_{\tau_j \in \tau_A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_{start} + (8 - \alpha(t_5, S'_I)), t_6 + 8) = 0. \tag{A.34}$$

   Also, $|t_{start} - t_6| + |t_{start} + (8 - \alpha(t_5, S'_I)) - (t_6 + 8)| = \alpha(t_5, S'_I)$. Lemma A.5 implies that $\alpha(t_5, S'_I) \leq 1$; therefore, X_6 must overlap with two jobs of τ_3 (otherwise, $\sum_{\tau_j \in \tau_{\text{example}} - \{\tau_6\}} W_j(S'_I, \pi_2, t_6, t_6 + 8) \leq 4$). Let r_1^3 and r_2^3 be the release times of the first and second jobs of τ_3 that overlaps with X_6. For $\sum_{\tau_j \in \tau_{\text{example}} - \{\tau_6\}} W_j(S'_I, \pi_2, t_6, t_6 + 8) > 4$ to be true, we must now show that τ_3 does not execute on processor π_2 in interval $[t_6, t_{start}]$ and $[t_{start} + (8 - \alpha(t_5, S'_I)), t_6]$ (i.e., there is not enough idle time on π_1 to accommodate the two jobs of τ_3). It must be that $|([r_1^3, r_1^3 + 2] \cup [r_2^3, r_2^3 + 2]) \cap X_{start}| \leq 4 - \alpha(t_5, S'_I)$ because $p_3 = 6$ and $d_3 = 2$. This means that together the two jobs of τ_3

overlap with at least $\alpha(t_5, S'_I)$ units of idle time on π_1. By the definition of S' and Equations A.33 and A.34, $W_3(S'_I, \pi_2, t_6, t_6 + 8) \leq 2 - \alpha(t_5, S'_I)$ because we may move at least $\alpha(t_5, S'_I)$ units of τ_3's execution to π_1. This implies $\sum_{\tau_j \in \tau_{\text{example}} - \{\tau_6\}} W_j(S'_I, \pi_2, t_6, t_6 + 8) \leq 4$ which contradicts the assumption. Thus, this case is impossible.

2. $t_{start} < t_6$:

   a) $t_5 > t_6 - 2$: Let $y \stackrel{\text{def}}{=} \max(t_6 - t_5, 0)$. Thus $t_5 + 6$ equals $t_6 - y + 6$. Since $[t_5, t_5 + 6] \subset [t_{start}, t_{start} + (8 - \alpha(t_5, S'_I))]$, it must be that $t_{start} + (8 - \alpha(t_5, S'_I)) \geq t_6 - y + 6$. By Lemma A.8, the amount of execution of $\tau_A \cup \{\tau_5\}$ in the interval $[t_{start}, t_{start} + (8 - \alpha(t_5, S'_I))]$ is maximized when $t_{start} + (8 - \alpha(t_5, S'_I))$ equals $t_6 - y + 6$. In this case, Lemma A.8 the earliest a job of $\tau_A \cup \{\tau_5\}$ can execute after $t_{start} + (8 - \alpha(t_5, S'_I))$ is at time $t_6 + 8 - y$. Therefore,

$$\sum_{\tau_j \in \tau_A \cup \{\tau_5\}} W_j(S'_I, \pi_1, t_{start} + (8 - \alpha(t_5, S'_I)), t_6 + 8) \leq y.$$

   We will now show that τ_3 can execute on processor π_2 for at most $2 - \alpha(t_5, S'_I)$ time units over the interval $[t_6, t_6 + 8]$. Observe that

$$|t_{start} + (8 - \alpha(t_5, S'_I)) - t_6| < 8 - \alpha(t_5, S'_I) - y \tag{A.35}$$

   and

$$\left|\left([r_1^3, r_1^3 + 2] \cup [r_2^3, r_2^3 + 2]\right) \cap [t_6, t_{start} + (8 - \alpha(t_5, S'_I))]\right| \leq 4 - \alpha(t_5, S'_I) - y. \tag{A.36}$$

By definition of S'_I and Equation A.36,

$$W_3(S'_I, \pi_2, t_6, t_{start} + (8 - \alpha(t_5, S'_I))) \leq 2 - \alpha(t_5, S'_I) - y. \tag{A.37}$$

τ_5 does not have a scheduling window over the interval of $[t_{start} + (8 - \alpha(t_5, S'_I)), t_6 + 8]$. So, by definition of S'_I and Equation A.35, τ_3 will execute for at most y on processor π_2 over the interval $[t_{start} + (8 - \alpha(t_5, S'_I)), t_6 + 8]$. Therefore,

$$W_3(S'_I, \pi_2, t_{start} + (8 - \alpha(t_5, S'_I)), t_6 + 8) \leq y. \tag{A.38}$$

By Equations A.31, A.37, and A.38 and the fact that τ_5 executes on processor π_2 for at most $\alpha(t_5, S'_I)$ time units implies

$$\begin{aligned} \textstyle\sum_{\tau_j \in \tau_{\text{example}} - \{\tau_6\}} W_j(S'_I, \pi_2, t_6, t_6 + 8) &= 2 + (2 - \alpha(t_5, S'_I)) + \alpha(t_5, S'_I) \\ &\leq 4. \end{aligned}$$

This contradicts our assumption; therefore, Case 2a is impossible.

b) $t_6 - 6 < t_5 \leq t_6 - 2$: Consider a modified schedule S''_I in which more of τ_5's

execution on processor π_2 is moved to the interval $[t_5, t_6)$.

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

$$S''_I(\pi_2, t) \stackrel{\text{def}}{=} \begin{cases} \varphi_5(I,t)\ , & \text{if } (t_5 \leq t < t_6) \text{ and} (S'_I(\pi_2,t) = \perp) \text{ and } (S'_I(\pi_1,t,\tau_5) = 0) \text{ and} \\ & (W_5(S''_I, \pi_2, r_5(I,t), t) < \alpha(t_5, S'_I)), \\ \perp\ , & \text{if } (S'_I(\pi_2,t,\tau_5) = 1) \text{ and} \\ & (W_5(S''_I, \pi_2, r_5(I,t), t) \geq \alpha(t_5, S'_I)), \\ S'_I(\pi_2,t)\ , & \text{otherwise} \end{cases} \tag{A.39}$$

It is easy to see that S''_I is valid, as we are only moving execution of τ_5 during τ_5's scheduling window. From the definition of S'_I and S''_I, the only times τ_5 does not execute in $[t_5, t_6)$ on processor π_2 is when:

i. π_1 is busy executing τ_5,

ii. π_2 is busy executing τ_2 or τ_3, or

iii. τ_5 has completed execution (i.e., τ_5 has executed for exactly $\alpha(t_5, S'_I)$ time units on π_2).

These cases taken together imply that in the interval $[t_5, t_6)$ for schedule S''_I processor π_2 must execute jobs of tasks in τ_B for at least $\alpha(t_5, S'_I)$ time units. More formally,

$$\sum_{\tau_j \in \tau_B} W_j(S''_I, \pi_2, t_5, t_6) \geq \alpha(t_5, S'_I).$$

Due to the fact that $t_6 - t_5 \geq 2$ and that $\text{EBF}(\tau_2, 10) = 2$ and $\text{EBF}(\tau_3, 10) = 2$,

$$\sum_{\tau_j \in \tau_{\text{example}} - \{\tau_6\}} W_j(S''_I, \pi_2, t_6, t_6 + 8) \leq 4.$$

We have defined a valid schedule S''_I that leave enough idle instants for τ_6 to execute entirely on processor π_2.

3. $t_{start} > t_6 + \alpha(t_5, S'_I)$:

   a) $t_5 < t_6 + 4$: Symmetric to Case 2a.

   b) $t_6 + 8 > t_5 \geq t_6 + 4$: Symmetric to Case 2b.

■

Bibliography

Abdelzaher, T., Sharma, V., and Lu, C. (2004). A utilization bound for aperiodic tasks and priority driven scheduling. *IEEE Transactions on Computers*, 53(3).

Albers, K. and Slomka, F. (2004). An event stream driven approximation for the analysis of real-time systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 187–195, Catania, Sicily. IEEE Computer Society Press.

Anderson, J. and Srinivasan, A. (2004). Mixed Pfair/ ERfair scheduling of asynchronous periodic tasks. *Journal of Computer And System Sciences*, 68(1):157–204.

Andersson, B., Abdelzaher, T., and Jonsson, J. (2003a). Global priority-driven aperiodic scheduling on multiprocessors. In *Proceedings of the Parallel and Distributed Processing Symposium*, Nice, France.

Andersson, B., Abdelzaher, T., and Jonsson, J. (2003b). Partitioned aperiodic scheduling on multiprocessors. In *Proceedings of the Parallel and Distributed Processing Symposium*, Nice, France.

Andersson, B. and Jonsson, J. (2000). Fixed-priority preemptive multiprocessor scheduling: To partition or not to partition. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications*, pages 337–346, Cheju Island, South Korea. IEEE Computer Society Press.

Andersson, B. and Jonsson, J. (2003). The utilization bounds of partitioned and pfair static-priority scheduling on multiprocessors are 50%. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 33–40, Porto, Portugal. IEEE Computer Society Press.

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

Baker, T. and Baruah, S. (2007). Schedulability analysis of multiprocessor sporadic task systems. In Son, S. H., Lee, I., and Leung, J. Y.-T., editors, *Handbook of Real-Time and Embedded Systems*. Chapman Hall/ CRC Press.

Baker, T. and Cirinei, M. (2006). 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 178–190, Rio de Janeiro. IEEE Computer Society Press.

Baker, T. and Shaw, A. (1989). The cyclic executive model and ada. *Real-Time Systems*, 1:7–26.

Baker, T. P. (2003). An analysis of deadline-monotonic schedulability on a multiprocessor. Technical Report TR-030201, Department of Computer Science, Florida State University.

Baker, T. P. (2005a). An analysis of EDF schedulability on a multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, 16(8):760–768.

Baker, T. P. (2005b). Comparison of empirical success rates of global vs. partitioned fixed-priority and EDF scheduling for hard real time. Technical Report TR-050601, Department of Computer Science, Florida State University.

Baker, T. P. (2006a). An analysis of fixed-priority schedulability on a multiprocessor. *Real-Time Systems: The International Journal of Time-Critical Computing*, 32(1–2):49–71.

Baker, T. P. (2006b). A comparison of global and partitioned EDF schedulability tests for multiprocessors. In *Proceeding of the International Conference on Real-Time and Network Systems*, pages 119–127, Poitiers, France.

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

Baruah, S. and Burns, A. (2006). Sustainable scheduling analysis. In *Proceedings of the IEEE Real-time Systems Symposium*, pages 159–168, Rio de Janeiro. IEEE Computer Society Press.

Baruah, S. and Carpenter, J. (2003). Multiprocessor fixed-priority scheduling with restricted interprocessor migrations. In *Proceedings of the EuroMicro Conference on Real-time Systems*, pages 195–202, Porto, Portugal. IEEE Computer Society Press.

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

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

Baruah, S. and Fisher, N. (2005a). Code-size minimization in multiprocessor real-time systems. In *Proceedings of the International Workshop on Parallel and Distributed Real-Time Systems*, pages 137a– 137a, Denver, Colorado.

Baruah, S. and Fisher, N. (2005b). The partitioned scheduling of sporadic real-time tasks on multiprocessor platforms. In *Proceedings of the Workshop on Compile/Runtime Techniques for Parallel Computing*, Oslo, Norway.

Baruah, S., Gehrke, J., and Plaxton, G. (1995). Fast scheduling of periodic tasks on multiple resources. In *Proceedings of the Ninth International Parallel Processing Symposium*, pages 280–288. IEEE Computer Society Press. Extended version available via anonymous ftp from `ftp.cs.utexas.edu`, as Tech Report TR–95–02.

Baruah, S., Howell, R., and Rosier, L. (1990a). Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Systems: The International Journal of Time-Critical Computing*, 2:301–324.

Baruah, S., Mok, A., and Rosier, L. (1990b). Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 182–190, Orlando, Florida. IEEE Computer Society Press.

Bertogna, M., Cirinei, M., and Lipari, G. (2005a). Improved schedulability analysis of EDF on multiprocessor platforms. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 209–218, Palma de Mallorca, Balearic Islands, Spain. IEEE Computer Society Press.

Bertogna, M., Cirinei, M., and Lipari, G. (2005b). New schedulability tests for real-time tasks sets scheduled by deadline monotonic on multiprocessors. In *Proceedings of the 9th International Conference on Principles of Distributed Systems*, pages 306–321, Pisa, Italy. Springer.

Bertogna, M., Fisher, N., and Baruah, S. (2006). Resource-locking durations in static-priority systems. Manuscript under review.

Bini, E. and Buttazzo, G. C. (2005). Measuring the performance of schedulability tests. *Real-Time Systems*, 30(1):129–154.

Calandrino, J., Anderson, J., and Baumberger, D. (2007). A hybrid real-time scheduling approach for large-scale multicore platforms. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, Pisa, Italy. IEEE Computer Society Press. To appear.

Calandrino, J., Leontyev, H., Block, A., Devi, U., and Anderson, J. (2006). Litmus^{RT}: A testbed for empirically comparing real-time multiprocessor schedulers. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 111–123, Rio de Janerio. IEEE Computer Society Press.

Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J., and Baruah, S. (2003). A categorization of real-time multiprocessor scheduling problems and

algorithms. In Leung, J. Y.-T., editor, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press LLC.

Chakraborty, S. (2003). *System-Level Timing Analysis and Scheduling for Embedded Packet Processors*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich. Available as Diss. ETH No. 15093.

Chakraborty, S., Erlebach, T., and Thiele, L. (2001). On the complexity of scheduling conditional real-time code. In *Proceedings of the 7th Workshop on Algorithms and Data Structures*, pages 38–49, Providence, RI. Springer Verlag.

Chen, D., Mok, A., and Baruah, S. (2000). Scheduling distributed real-time tasks in the dgmf model. In *IEEE Real-Time Technology and Applications Symposium*, pages 14–22. IEEE.

Deng, Z. and Liu, J. (1997). Scheduling real-time applications in an Open environment. In *Proceedings of the Eighteenth Real-Time Systems Symposium*, pages 308–319, San Francisco, CA. IEEE Computer Society Press.

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

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

Devi, U. (2003). An improved schedulability test for uniprocessor periodic task systems. In *Proceedings of the EuroMicro Conference on Real-time Systems*, Porto, Portugal. IEEE Computer Society Press.

Devi, U. (2006). *Soft Real-Time Scheduling on Multiprocessors*. PhD thesis, Department of Computer Science, The University of North Carolina at Chapel Hill.

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

Fisher, N., Anderson, J., and Baruah, S. (2005). Task partitioning upon memory-constrained multiprocessors. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 416–421, Hong Kong. IEEE Computer Society Press.

Fisher, N. and Baruah, S. (2005a). A fully polynomial-time approximation scheme for feasibility analysis in static-priority systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 117–126, Palma de Mallorca, Balearic Islands, Spain. IEEE Computer Society Press.

Fisher, N. and Baruah, S. (2005b). A polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. In *Proceedings of the 13th International Conference on Real-Time Systems*, pages 233–249, Paris, France.

Fisher, N., Bertogna, M., and Baruah, S. (2007a). Resource-locking durations in edf-scheduled systems. In *13th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 91–100. IEEE.

Fisher, N., Nguyen, T. H. C., Goossens, J., and Richard, P. (2007b). Parametric polynomial-time algorithms for computing response-time bounds for static-priority tasks with release jitters. In *Proceedings of the International Conference on Real-Time Computing Systems and Applications*, Daegu, Korea. IEEE Computer Society Press. To appear.

Ghazalie, T. M. and Baker, T. P. (1995). Aperiodic servers in a deadline scheduling environment. *Real-Time Systems*, 9.

Ha, R. and Liu, J. W. S. (1994). Validating timing constraints in multiprocessor and distributed real-time systems. In *Proceedings of the 14th IEEE International Conference on Distributed Computing Systems*, pages 162–171, Los Alamitos. IEEE Computer Society Press.

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

Horn, W. (1974). Some simple scheduling algorithms. *Naval Research Logistics Quarterly*, 21:177–185.

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

Johnson, D. (1974). Fast algorithms for bin packing. *Journal of Computer and Systems Science*, 8(3):272–314.

Johnson, D. S. (1973). *Near-optimal Bin Packing Algorithms*. PhD thesis, Department of Mathematics, Massachusetts Institute of Technology.

Kalyanasundaram, B. and Pruhs, K. (2000). Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643.

Kirsch, C., Sanvido, M., Henzinger, T., and Pree, W. (2002). A giotto-based helicopter control system. In *Proceedings of the Second International Workshop on Embedded Software (EMSOFT)*, volume 2491 of *Lecture Notes in Computer Science*, pages 46–60. Springer-Verlag.

Kolmogorov, A. N. and Fomin, S. V. (1970). *Introductory Real Analysis*. Dover Publications, Inc., New York.

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

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

Liu, J. W. S. (2000). *Real-Time Systems*. Prentice-Hall, Inc., Upper Saddle River, New Jersey 07458.

Lopez, J. M., Diaz, J. L., and Garcia, D. F. (2004). Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems: The International Journal of Time-Critical Computing*, 28(1):39–68.

Lopez, J. M., Garcia, M., Diaz, J. L., and Garcia, D. F. (2000). Worst-case utilization bound for EDF scheduling in real-time multiprocessor systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 25–34, Stockholm, Sweden. IEEE Computer Society Press.

Lundberg, L. and Lennerstad, H. (2003). Global multiprocessor scheduling of aperiodic tasks using time-independent priorities. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pages 170–180. IEEE Computer Society Press.

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

Mok, A. K. and Chen, D. (1996). A multiframe model for real-time tasks. In *Proceedings of the 17th Real-Time Systems Symposium*, Washington, DC. IEEE Computer Society Press.

Mok, A. K. and Chen, D. (1997). A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645.

Murty, K. G. (1983). *Linear Programming*. John Wiley & Sons, Inc., New York.

Oh, D.-I. and Baker, T. P. (1998). Utilization bounds for N-processor rate monotone scheduling with static processor assignment. *Real-Time Systems: The International Journal of Time-Critical Computing*, 15:183–192.

Park, D.-W., Natarajan, S., Kanevsky, A., and Kim, M. J. (1995). A generalized utilization bound test for fixed-priority real-time scheduling. In *2nd International Workshop on Real-Time Computing Systems and Applications*, pages 68–72, Washington, DC, USA. IEEE Computer Society.

Phillips, C. A., Stein, C., Torng, E., and Wein, J. (1997). 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.

Phillips, C. A., Stein, C., Torng, E., and Wein, J. (2002). Optimal time-critical scheduling via resource augmentation. *Algorithmica*, 32(2):163–200.

Richard, P., Goossens, J., and Fisher, N. (2007). Approximate feasibility analysis and response-time bounds of static-priority tasks with release jitters. In *Proceedings of the 15th International Conference on Real-Time and Network Systems*, pages 105–112, Nancy, France.

Ripoll, I., Crespo, A., and Mok, A. K. (1996). Improvement in feasibility testing for real-time tasks. *Real-Time Systems: The International Journal of Time-Critical Computing*, 11:19–39.

Saez, S., Vila, J., and Crespo, A. (1998). Using exact feasibility tests for allocating real-time tasks in multiprocessor systems. In *Proceedings of the Tenth EuroMicro Workshop on Real-time Systems*, pages 53–60, Berlin, Germany.

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

Srinivasan, A. and Baruah, S. (2002). Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84(2):93–98.