Compositional Analysis Techniques For Multiprocessor Soft Real-Time Scheduling

Hennadiy Leontyev

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
2010

Approved by,

Prof. James H. Anderson

Prof. Sanjoy Baruah

Prof. Kevin Jeffay

Prof. Ketan Mayer-Patel

Prof. Jasleen Kaur

Prof. Samarjit Chakraborty

ABSTRACT

HENNADIY LEONTYEV: Compositional Analysis Techniques For Multiprocessor Soft Real-Time Scheduling.

(Under the direction of Prof. James H. Anderson)

The design of systems in which timing constraints must be met (real-time systems) is being affected by three trends in hardware and software development. First, in the past few years, multiprocessor and multicore platforms have become standard in desktop and server systems and continue to expand in the domain of embedded systems. Second, real-time concepts are being applied in the design of general-purpose operating systems (like Linux) and attempts are being made to tailor these systems to support tasks with timing constraints. Third, in many embedded systems, it is now more economical to use a single multiprocessor instead of several uniprocessor elements; this motivates the need to share the increasing processing capacity of multiprocessor platforms among several applications supplied by different vendors and each having different timing constraints in a manner that ensures that these constraints were met. These trends suggest the need for mechanisms that enable real-time tasks to be bundled into multiple components and integrated in larger settings.

There is a substantial body of prior work on the multiprocessor schedulability analysis of real-time systems modeled as periodic and sporadic task systems. Unfortunately, these standard task models can be pessimistic if long chains of dependent tasks are being analyzed. In work that introduces less pessimistic and more sophisticated workload models, only partitioned scheduling is assumed so that each task is statically assigned to some processor. This results in pessimism in the amount of needed processing resources.

In this dissertation, we extend prior work on multiprocessor soft real-time scheduling and construct new analysis tools that can be used to design component-based soft real-time systems. These tools allow multiprocessor real-time systems to be designed and analyzed for which standard workload and platform models are inapplicable and for which state-of-the-art uniprocessor and multiprocessor analysis techniques give results that are too pessimistic.

ACKNOWLEDGMENTS

My dissertation and graduate school career would not have been possible without the help of many people. First, I would also like to thank my dissertation committee: James Anderson, Sanjoy Baruah, Kevin Jeffay, Ketan Mayer-Patel, Jasleen Kaur, and Samarjit Chakraborty, for the feedback they have provided during my work. Especially, I am grateful to my advisor, Jim Anderson, who patiently guided me through research and writing over these five years. I cannot imagine a better advisor.

I would also like to thank the UNC Department of Computer Science as a whole for its positive and friendly environment. Due to some great teachers here, I learned more about computer science than I had learned during my previous five years as a student. I owe much to the many colleagues with whom I have published over the years: Uma Devi, Björn Brandenburg, John Calandrino, Aaron Block. Also, I owe many thanks to my other real-time colleagues: Nathan Fisher and Cong Liu. I wished I had written a paper with you.

I would like to thank people at places where I did my two summer internships in 2007 and 2009: the School of Computing at National University of Singapore and AT&T Labs Research. My collaborator Theodore Johnson at AT&T deserves a large amount of credit for showing that my research can really have a big impact.

Finally, I want to thank my wife. Maria, you are the most wonderful wife I could have asked for. Without you, I would not be able to finish this dissertation and graduate. Thank you for your unconditional love, continuous support, and patience. I love you so much.

Thanks again, everyone. Enjoy the reading.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CA	Container-Aware Scheduling
EDF	Earliest Deadline First
EDL	Earliest Deadline Last
EDZL	Earliest Deadline Zero Laxity
EPDF	Earliest Pseudo-Deadline First
FIFO	First-In-First-Out
FP	Fixed-Priority
GEDF	Global EDF
HRT	Hard Real-Time
HS	Hard-Soft Scheduling
LLF	Least Laxity First
LLREF	Least Local Remaining Execution First
NPGEDF	Non-preemptive Global EDF
PEDF	Partitioned EDF
PS	Processor-Sharing
RM	Rate-Monotonic
SRT	Soft Real-Time
TDMA	Time-Division Multiple Access

Chapter 1

Introduction

The goal of this dissertation is to extend prior work on multiprocessor real-time scheduling to enable soft real-time schedulability theory to meet the expectations of system designers. The particular focus of this work is sets of real-time tasks that need to be integrated as components in larger settings. Such settings include stream-processing (multimedia) applications, systems where computing resources are shared among multiple real-time applications, embedded systems, etc. Prior to the research in this dissertation, scheduling in multiprocessor soft real-time systems has been mainly considered in standalone contexts. In this dissertation, we extend prior work on multiprocessor soft real-time scheduling and construct new analysis tools that can be used to design component-based soft real-time systems. Further, we present novel validation procedures for several well-known scheduling algorithms that allow heterogeneous real-time constraints to be tested in a uniform fashion.

To motivate the need for compositional analysis, we start with a brief introduction to real-time systems. Next, we present the system model that is assumed in this dissertation. We then briefly review prior work on multiprocessor soft real-time scheduling and compositional analysis and state the thesis of this dissertation. Finally, we summarize this dissertation's contributions and give an outline for the remainder of the dissertation.

1.1 What is a Real-Time System?

As opposed to many computer systems, real-time systems have *timing requirements* that must be satisfied. Thus, a real-time system has a dual notion of correctness: the programs comprising such a system should not only produce results in accordance with their functional specifications but should also have these computations finish within specified time frames. The latter property is called *temporal correct-*

ness. Embedded systems such as automotive controllers and medical devices, some multimedia software, radar signal-processing, and tracking systems are the examples of real-time systems.

Timing constraints are often specified in terms of *deadlines* for activities. Based on the cost of failure associated with not meeting them, deadlines in real-time systems can be broadly classified as either *hard* or *soft*. A hard real-time deadline is one whose violation can lead to disastrous consequences such as loss of life or a significant loss to property. Industrial process-control systems and robots, controllers for automotive systems, and air-traffic-control systems are some examples of systems with hard deadlines. In contrast, a soft deadline is less critical; hence, soft deadlines can occasionally be violated. However, such violations are not desirable, either, as they may lead to degraded quality of service. For example, in an HDTV player, a new video frame must be created and displayed every 33 milliseconds. If a frame is not processed on time (a deadline is missed), then there may be a perceptible disruption in the displayed video. Another example of a soft real-time application is a real-time data warehouse. Such a system periodically gathers data across a large-scale computer network and analyzes the data in order to identify network performance problems (Golab et al., 2009). As long as most deadlines are met, network problems can be properly detected and handled as they happen. Many multimedia systems and virtual-reality systems also have soft real-time constraints (Block, 2008; Bennett, 2007; Bennett and McMillan, 2005; Vallidis, 2002).

For a real-time system, it should be possible to ensure that all timing requirements can always be met under the assumptions made concerning the system. In other words, the system should be *predictable*. Ensuring *a priori* that timing requirements are met is the core of real-time systems theory and the subject of concentration of this dissertation. In order to make such predictions, for complex real-time systems in which global (resource-efficient) scheduling algorithms are used, appropriate analysis tools are yet to be developed. This motivates the research addressed in this dissertation as explained in the next section in greater detail.

1.2   Motivation

The main goal of this dissertation is to bridge the gap between the current state-of-the-art in multiprocessor soft real-time scheduling and real-world needs. Such needs are being impacted by three trends in hardware and software development.

First, general-purpose operating systems (OSs) are becoming more "real-time capable" via the introduction of "real-time" features such as high-resolution timers, short non-preemptable code segments,

and in-kernel priority-inheritance mechanisms (e.g., the RT-PREEMPT patch for the Linux kernel (RTp, 2009)). This trend has been driven by a growth in applications with timing constraints that developers wish to host on such systems.

Second, new features are being introduced to support "co-hosted" applications. Though general-purpose OSs are typically used to run several applications simultaneously, in some situations, one application may occupy all available system resources and make the entire system unresponsive. To prevent such behaviors, strong isolation mechanisms known as *application containers* have been introduced in Linux (LVS, 2007; Eriksson and Palmroos, 2007; Lessard, 2003). Containers are an abstraction that allows different application groups to be isolated from one another (mainly, by providing different name spaces to different application groups for referring to programs, files, etc.). Containers are seen as a lightweight way to achieve many of the benefits provided by virtualization without the expense of running multiple OSs. For example, quotas on various system resources such as processor time, memory size, network bandwidth, etc., can be enforced for encapsulated applications.

Third, these OS-related developments are happening at a time when multicore processors are now in widespread use. Additionally, reasonably-priced "server class" multiprocessors have been available for some time now. One such machine can provide many functions, including soft real-time applications like HDTV streaming and interactive video games, thus serving as a *multi-purpose home appliance* (Intel Corporation, 2006). The spectrum of settings where multicore architectures are being used even includes handheld devices. The resulting increase in processing power on such devices enables MPEG video encoding/decoding software to be deployed on them. These hardware-related developments are profound, because they mean that multiprocessors are now a "common-case" platform that software designers must deal with.

As the above discussion suggests, recent changes made in common hardware and OS architectures motivate the problem of sharing the processing capacity of one multiprocessor machine among multiple real-time applications in a *predictable* manner. Deploying multiple real-time applications on a multiprocessor platform can be seen as an aspect of the larger issue of *composability*. The increasing complexity and heterogeneity of modern embedded platforms have led to a growing interest in compositional modeling and analysis techniques (Richter et al., 2003; Chakraborty et al., 2003, 2006). In devising such techniques, the goal is not only to analyze the individual components of a platform in isolation, but also to compose different analysis results to estimate the timing and performance characteristics of the entire platform. Such analysis should be applicable even if individual processing and communication elements implement different scheduling/arbitration policies, have different interfaces, and are supplied

by different vendors. These complicating factors often cause standard workload models and analysis techniques to lead to overly pessimistic results. *To enable efficient system design solutions and to reduce design and verification costs, existing compositional frameworks need to be extended so that soft real-time workloads can be efficiently supported on multiprocessor platforms.*

Unlike most prior related efforts (see Chapter 2), we are mainly interested in supporting soft timing constraints. There is growing awareness in the real-time-systems community that, in many settings, soft constraints are far more common than hard constraints (Rajkumar, 2006). If hard constraints do exist, then ensuring them *efficiently* on most multiprocessor platforms is problematic for several reasons. First, various processor components such as caches, instruction pipelines, and branch-prediction mechanisms make it virtually impossible to estimate worst-case execution times of programs accurately. (While execution times are needed to analyze soft real-time systems as well, less-accurate empirically-derived costs often suffice in such systems.) Second, while there is much interest in tailoring OSs like Linux to support soft real-time workloads, such OSs are not real-time operating systems and thus cannot be used to support "true" hard timing constraints.

Real-time programs are typically implemented as a collection of threads or tasks. A scheduling algorithm determines which task(s) should be running at any time. A task model describes the parameters of a set of real-time tasks and their timing constraints. On the other hand, a resource model describes the resources available on a hardware platform for executing tasks. The most basic analysis of a real-time system involves running validation tests, which determine whether a real-time system's timing constraints will be met if a specified scheduling algorithm is used.

In the next section, we describe one of the real-time task models studied in this dissertation. In Section 1.4, a resource model is presented. In Section 1.5, we present some important scheduling algorithms and schedulability tests for them (more algorithms and tests are discussed in detail in Chapter 2).

1.3   Real-Time Task Model

In this section, we describe the sporadic task model and the timing constraints under it. Later, in Section 1.6, we describe a generalization of the sporadic task model called the streaming task model, which circumvents some of the limitations of the sporadic task model and is used for the analysis of component-based systems.

1.3.1 Sporadic Task Model

Many real-time systems consist of one or more sequential segments of code, called *tasks*, each of which is invoked (or released) repeatedly, with each invocation needing to complete within a specified amount of time. Tasks can be invoked in response to events in the external environment that the system interacts with, events in other tasks, or the passage of time as determined by using timers. Each invocation of a task is called a *job* of that task, and unless otherwise specified, a task is long-lived, and can be invoked an infinite number of times, i.e., can generate jobs indefinitely.

In this dissertation, we consider a set of n sequential tasks $\tau = \{T_1, T_2, \ldots, T_n\}$. Associated with each task T_i are three parameters, e_i, p_i, and D_i: e_i gives the *worst-case execution time* (WCET) of any job of T_i, which is the maximum time such a job can execute on a dedicated processor; $p_i \geq e_i$, called the *period* of T_i, is the minimum time between consecutive job releases; and $D_i \geq e_i$, called the *relative deadline* of T_i, denotes the amount of time within which each job of T_i should complete execution after its release.

The jth job of T_i, where $j \geq 1$, is denoted $T_{i,j}$. A task's first job may be released at any time $t \geq 0$. The *arrival* or *release time* of job $T_{i,j}$ is denoted $r_{i,j}$ and its (absolute) deadline $d_{i,j}$ is defined as $r_{i,j} + D_i$. The completion time of $T_{i,j}$ is denoted $f_{i,j}$ and $f_{i,j} - r_{i,j}$ is called its *response time*. Task T_i's maximum response time is defined as $\max_{j \geq 1}(f_{i,j} - r_{i,j})$. The execution time of job $T_{i,j}$ is denoted $e_{i,j}$.

For each job $T_{i,j}$, we define an *eligibility time* $\epsilon_{i,j}$ such that $\epsilon_{i,j} \leq r_{i,j}$ and $\epsilon_{i,j-1} \leq \epsilon_{i,j}$. The eligibility time of $T_{i,j}$ denotes the earliest time when it may be scheduled. A job $T_{i,j}$ is said to be *early-released* if $\epsilon_{i,j} < r_{i,j}$. An unfinished job $T_{i,j}$ is said to be *eligible* at time t if $t \geq \epsilon_{i,j}$. The early-release task model was considered in prior work on Pfair scheduling (Anderson and Srinivasan, 2004). As shown later in Example 2.3 in Section 2.1.2, allowing early releases can reduce job response times.

If $D_i = p_i$ (respectively, $D_i \leq p_i$) holds, then T_i and its jobs are said to have *implicit deadlines* (respectively, *constrained deadlines*). A sporadic task system in which $D_i = p_i$ (respectively, $D_i \leq p_i$) holds for each task is said to be an *implicit-deadline system* (respectively, *constrained-deadline system*). In an *arbitrary-deadline system*, there are no constraints on relative deadlines and periods. For brevity, we often use the notation $T_i(e_i, p_i, D_i)$ to specify task parameters in constrained- and arbitrary-deadline systems and $T_i(e_i, p_i)$ in implicit-deadline systems.

In this dissertation, we consider schedules in which jobs are allowed to execute after their deadlines. If a job $T_{i,j}$ misses its deadline in a schedule $\mathcal{S}$, then it is said to be *tardy* and the extent of the miss is its *tardiness*. More generally, the *tardiness* of job $T_{i,j}$ in schedule $\mathcal{S}$ is defined as $tardiness(T_{i,j}, \mathcal{S}) =$
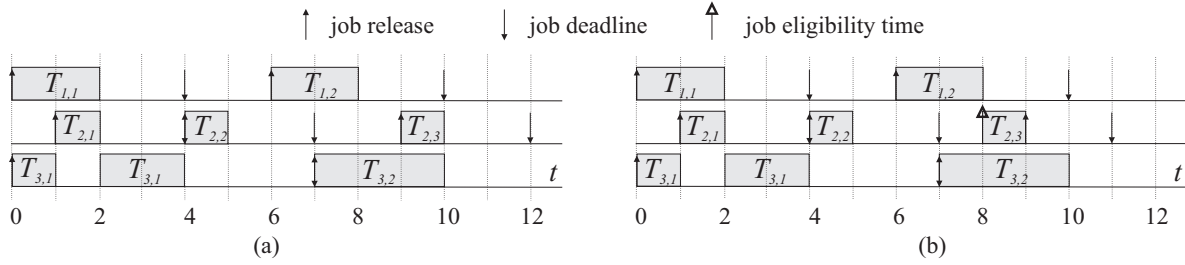
Figure 1.1: Example schedules of a sporadic task system from Example 1.1.

$\max(0, f_{i,j} - d_{i,j})$, and the *tardiness* of task T_i in schedule $\mathcal{S}$ is defined as $tardiness(T_i, \mathcal{S}) = \max_{j \geq 1}(tardiness(T_{i,j}, \mathcal{S}))$.

Because T_i is sequential, its jobs may not execute on multiple processors at the same time, i.e., parallelism is forbidden even if deadlines are missed. Further, a tardy job does not delay the releases of later jobs of the same task.

A task with the characteristics as described is referred to as a *sporadic task* and a task system composed of sporadic tasks is referred to as a *sporadic task system*. A *periodic task* T_i is a special case of a sporadic task in which consecutive job releases are separated by exactly p_i time units, and a task system whose tasks are all periodic is referred to as a *periodic task system*. A periodic task system is called *synchronous* if all tasks release their first jobs at the same time, and *asynchronous*, otherwise.

Example 1.1. An example sporadic task system with two implicit-deadline sporadic tasks $T_1(2, 4)$ and $T_2(1, 3)$ and one periodic task $T_3(3, 7)$ running on two processors is shown in Figure 1.1(a). Figure 1.1(b) shows the same task system except that job $T_{2,3}$ is released early by one time unit. In this example, we assume that jobs of T_1 have higher priority than those of T_2 and T_3. In the rest of the dissertation, up-arrows will denote job releases and down-arrows will denote job deadlines (if any).

Definition 1.1. The *utilization* of sporadic task T_i is defined as $u_i = e_i / p_i$, and the *utilization of the task system* τ as $U_{sum}(\tau) = \sum_{T_i \in \tau} u_i$.

The utilization of T_i is the maximum fraction of time on a dedicated processor that can be consumed by T_i's jobs over an interval during which a large number of T_i's jobs are released. In Example 1.1, task T_1 can consume up to half of the available processing time on a dedicated processor.

1.3.2 Hard vs. Soft Timing Constraints

A sporadic task T_i is called a *hard real-time* (Hard Real-Time (HRT)) *task* if no job deadline should be missed, i.e., $tardiness(T_i, \mathcal{S}) = 0$ is required. A system solely comprised of HRT tasks is called a *hard real-time* (HRT) *system*.

Alternatively, if, for task T_i, deadline misses are allowed, then T_i is called a *soft real-time* (Soft Real-Time (SRT)) *task*. The system containing one or more SRT tasks is called a *soft real-time* (SRT) *system*. Because jobs in SRT systems may miss deadlines occasionally, there is no single notion of SRT correctness. Some possible notions of SRT correctness include: bounded deadline tardiness (i.e., each job completes within some bounded time after its deadline) (Devi, 2006); a specified percentage of deadlines must be met (Atlas and Bestavros, 1998); and m out of every k consecutive jobs of each task complete before their deadlines (Hamdaoui and Ramanathan, 1995). In this dissertation, we are primarily concerned with HRT systems and SRT systems with bounded deadline tardiness. Bounded tardiness is important because each task with bounded tardiness can be guaranteed in the long run to receive processor time proportional to its utilization.

With HRT and SRT correctness defined as above, HRT correctness is simply a special case of SRT correctness. In both cases, we are concerned with whether a task's response time occurs within a specified bound. If a task's maximum response time is required to be at most its relative deadline, then that task is a HRT task. If it is required to be at most the relative deadline plus the maximum allowed tardiness, then that task is a SRT task.

In Chapters 3 and 4, we will specify timing requirements in terms of deadlines and tardiness. In Chapter 5, we will specify timing constraints in terms of maximum response times.

1.4 Resource Model

In this dissertation, we consider real-time task systems running on a platform comprised of a set of $m \geq 2$ identical unit-speed processors. Such a platform is called an *identical multiprocessor* platform. In this setting, all processors have the same characteristics, including uniform access times (in the absence of contention) to memory. Later, in Chapter 3, we also discuss how some of the results of this dissertation can be applied to *uniform multiprocessor* platforms, in which processors can have different speeds, i.e., different processors may execute instructions at different rates. Unless stated otherwise, in this dissertation, we assume that the platform is an identical multiprocessor.

In identical multiprocessor platforms, a memory access is accomplished by the use of a centralized shared memory. This type of multiprocessor is commonly referred to as a symmetric shared-memory multiprocessor (SMP) (see Figure 1.2(a) for an illustration). Each processor can have one or more levels of caches (instruction, data, and unified) to reduce memory access times. We assume that every task can execute on every processor except that it cannot occupy more than one processor at any time. If a
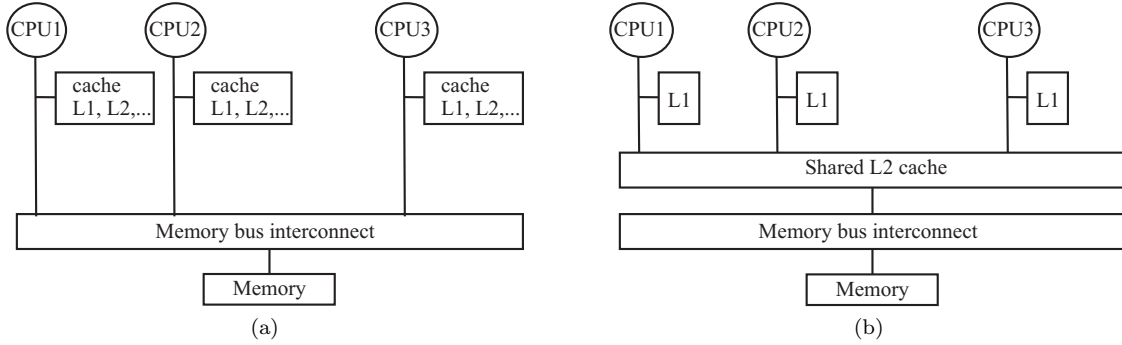
Figure 1.2: Symmetric multiprocessor architecture **(a)** without and **(b)** with a shared cache.

job (task) executes on different processors at different times, then we say that that job (task) *migrates*. When a job migrates, it may be necessary to load task-related instructions and data into a local cache. One of the ways to lower migration overheads is to restrict the execution of a task or a job to one or a subset of processors. Another way is to use a multicore architecture with shared caches. As the name suggests, the multicore chip has several processing cores on one die, which reduces power consumption and production costs. In addition, different cores may share a cache at some level as illustrated in Figure 1.2(b). Shared caches may reduce migration overheads, if task-related data and instructions do not need to be loaded from memory after a migration. Task preemptions, context switches, task migrations, and scheduler activity are system overheads that take processor time from the task system. It is not possible to predict the behavior of the system without accounting for these overheads. This problem is exacerbated in a platform with shared caches: due to cache interference, each individual job's execution time will depend on the job set being currently scheduled. Commonly, overheads are accounted for by charging each external activity (e.g., a preemption, migration, or scheduler invocation) to a unique job, and the WCET of each task is inflated by the maximum cumulative time required for all the external activities charged to any of its jobs. Throughout this dissertation, we will assume that system overheads are included in the WCETs of tasks using efficient charging methods (Devi, 2006). The WCET of a task is therefore dependent on the implementation platform, application characteristics, and the scheduling algorithm.

As noted in Section 1.2, the processing capacity of a multiprocessor platform often needs to be shared among multiple task systems in a predictable manner. In this case, from the standpoint of a single task system, the full capacity of one or more processors is not available to its constituent tasks. We assume that such capacity restrictions are specified using *service (supply) functions* (Bini et al., 2009b; Chakraborty et al., 2003; Mok et al., 2001).
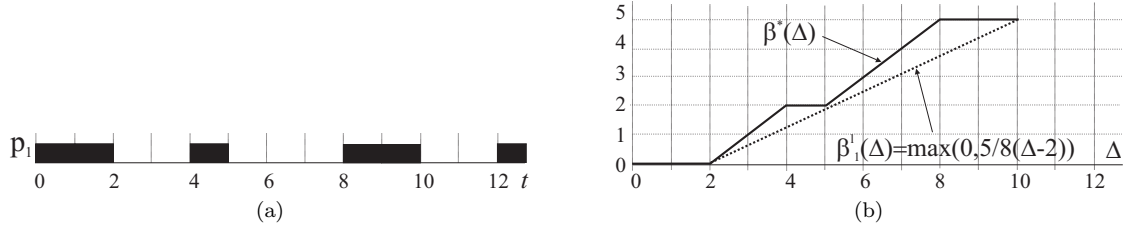
Figure 1.3: **(a)** Unavailable time instants and **(b)** service functions for processor 1 (denoted P_1) in Example 1.2.

Definition 1.2. The minimum guaranteed time that processor h can provide to τ in any time interval of length $\Delta \geq 0$ is characterized by the service function

$$\beta_h^l(\Delta) = \max(0, \widehat{u_h} \cdot (\Delta - \sigma_h)), \tag{1.1}$$

where $\widehat{u_h} \in (0, 1]$ and $\sigma_h \geq 0$.

In the above definition, $\widehat{u_h}$ is the total long-term utilization available to the tasks in τ on processor h and σ_h is the maximum length of time when the processor can be unavailable. These parameters are similar to those in the bounded delay model (Mok et al., 2001) and multi-supply function abstraction (Bini et al., 2009b). We require $\widehat{u_h}(\Delta)$ and σ_h to be specified for each h. Note that, if (unit-speed) processor h is fully available to the tasks in τ, then $\beta_h^l(\Delta) = \Delta$.

Example 1.2. Consider a system with a processor that is not fully available. The availability pattern, which repeats every eight time units, is shown in Figure 1.3(a); intervals of unavailability are shown as black regions. For processor 1, the minimum amount of time that is guaranteed to τ over any interval of length Δ is zero if $\Delta \leq 2$, $\Delta - 2$ if $2 \leq \Delta \leq 4$, and so on. Figure 1.3(b) shows the minimum amount of time $\beta^*(\Delta)$ that is available on processor 1 for soft real-time tasks over any interval of length Δ. It also shows a service curve $\beta_1(\Delta) = \max(0, \widehat{u_1}(\Delta - \sigma_1))$, where $\widehat{u_1} = \frac{5}{8}$ and $\sigma_1 = 2$, which bounds $\beta^*(\Delta)$ from below. $\beta_1(\Delta)$ can be used to reflect the minimum service guarantee for processor 1.

There exist many settings in which individual processor service functions are not known and a lower bound on the cumulative available processor time is provided instead. In this case, we let $\mathcal{B}(\Delta)$ be a lower-bound on the cumulative processor time available over any interval of length Δ. In Chapters 3 and 4 we assume that individual processor supplies are known and, in Chapter 5, we assume that the cumulative processor supply is known.

9

1.5 Real-Time Scheduling Algorithms and Tests

The ultimate goal of real-time systems analysis is guaranteeing temporal correctness. That is verifying *a priori* that no job deadline is ever missed or, if deadlines are allowed to be missed, then these misses are by no more than certain amount of time. Temporal correctness depends on how jobs are scheduled. A *scheduling algorithm* is used at runtime to determine which job to run next on the available processors.

Definition 1.3. A task system τ is *concrete* if the release and eligibility times of all of its jobs are specified and is *non-concrete* otherwise.

The task set considered in Example 1.1 is a non-concrete task system, while the schedules considered in this example are produced by two concrete instantiations of τ with different eligibility times for $T_{2,3}$.

In the real-time systems literature, a concrete task system τ is *feasible* on a given platform if there exists a schedule in which no job deadline is missed. A non-concrete task system τ is feasible on a given platform if every concrete instantiation of τ is feasible.

A HRT system τ is called *schedulable* under scheduling algorithm $\mathcal{A}$ on a given platform if no deadline is missed in the schedule produced by $\mathcal{A}$ for any concrete instantiation of τ. Alternatively, a SRT system τ is schedulable under $\mathcal{A}$ if the maximum task tardiness is bounded. Often, tardiness bounds are *specified* by system designers. Let Θ_i be the maximum allowed tardiness for task T_i. (Note that if $\Theta_i = 0$ for each task T_i, then the system is HRT.) In this case, τ is schedulable if these specified tardiness bounds are not exceeded.

Associated with a scheduling algorithm is a procedure for verifying schedulability called a *schedulability test*. In the rest of the section, we briefly describe the earliest-deadline-first (Earliest Deadline First (EDF)) scheduling algorithm for uniprocessor and multiprocessor platforms and schedulability results for it when considering implicit-deadline task systems. Other important scheduling algorithms, their associated schedulability tests, and schedulability tests for EDF for constrained- and arbitrary-deadline sporadic task systems are discussed in detail in Chapter 2. In the discussion below, we assume that all processors are fully available.

1.5.1 Uniprocessor Scheduling

For uniprocessor systems, every feasible task system can be scheduled by the preemptive EDF algorithm, which gives higher priority to jobs with smaller deadlines, so that all deadlines are met. This means that EDF is *optimal* for uniprocessor scheduling.

For an implicit-deadline task system τ, all deadlines can be met iff $U_{sum}(\tau) \leq 1$ (Liu and Layland, 1973). In contrast, if $U_{sum}(\tau) > 1$, then some tasks in τ have unbounded deadline tardiness in certain concrete instantiations of τ (e.g., when job releases are periodic). Therefore, the notions of HRT and SRT schedulability are the same for implicit-deadline systems under uniprocessor EDF.

1.5.2 Partitioned Multiprocessor Scheduling

Similarly to uniprocessor scheduling, under multiprocessor scheduling, an implicit-deadline task system τ is *feasible* on m processors iff $U_{sum}(\tau) \leq m$ (Anderson and Srinivasan, 2000; Baruah et al., 1996). If τ is schedulable using an algorithm $\mathcal{A}$ on m' processors, then the difference $m' - U_{sum}(\tau)$ is called the *utilization loss*. We would like to minimize such loss while still be able to satisfy all timing requirements.

Most multiprocessor scheduling algorithms can be classified as either partitioned or global (or some combination thereof). In *partitioned* algorithms, each task is permanently assigned to a specific processor and each processor independently schedules its assigned tasks using a uniprocessor scheduling algorithm. In *global* scheduling algorithms, tasks are scheduled from a single priority queue and may migrate among processors.

The advantage of partitioned schedulers is that they enable uniprocessor schedulers to be used (on each processor) and usually have low migration/preemption costs. The disadvantage of partitioned schedulers is that they may require more processors to schedule a task system when compared to global schedulers (as we will see later in this section). In this section, we consider the partitioned EDF (Partitioned EDF (PEDF)) scheduling algorithm.

Because uniprocessor EDF is optimal, for implicit-deadline task systems, it suffices to construct a partition of τ into the m subsets $\{\tau_k\}$ such that, for each k, $U_{sum}(\tau_k) \leq 1$. This partitioning problem is related to the NP-complete bin-packing problem and becomes even more difficult for restricted- and arbitrary-deadline systems. For these task systems, some sufficient schedulability tests have been developed for partitioned EDF and static-priority scheduling (Baruah and Fisher, 2006, 2007). Unfortunately, not all task sets can be successfully partitioned. In general, an implicit-deadline task system τ with utilization $U_{sum}(\tau)$ could require up to $\lceil 2 \cdot U_{sum}(\tau) - 1 \rceil$ processors in order to be schedulable using PEDF. (This and a more accurate bound, which depends on the maximum per-task utilization, are given in (Lopez et al., 2004).) In other words, up to half of the total available processor time can be unused under PEDF in the long run.
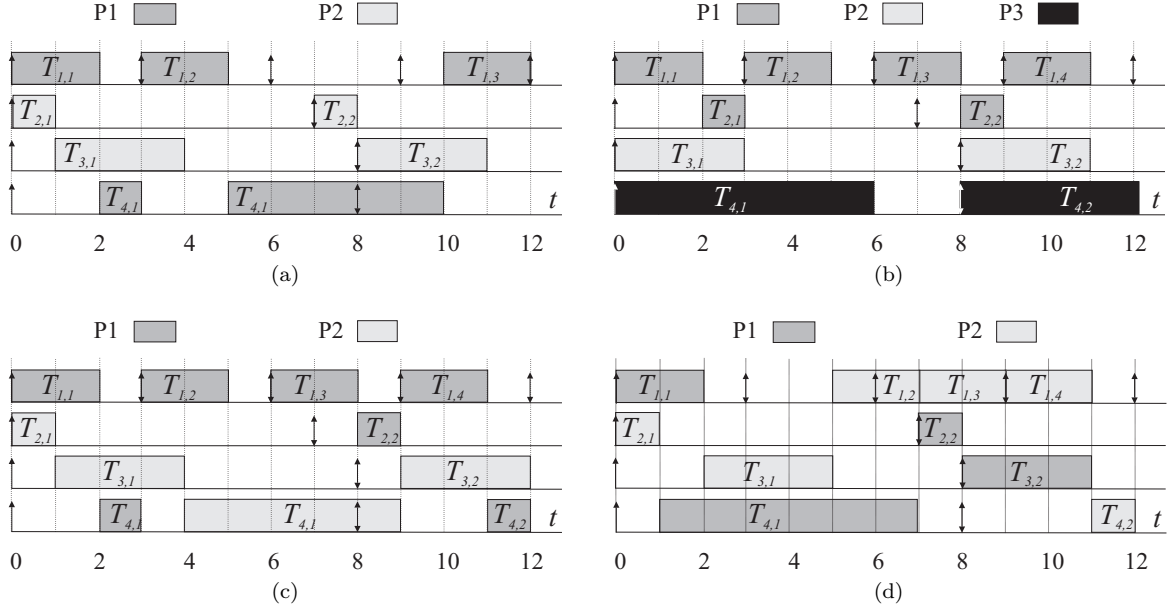
Figure 1.4: **(a)** Two- and **(b)** three-processor PEDF schedules in Example 1.3. **(c)** Preemptive and **(d)** nonpreemptive GEDF schedules in Example 1.4.

Example 1.3. Consider an implicit-deadline task system $\tau = \{T_1(2,3), T_2(1,7), T_3(3,8), T_4(6,8)\}$, which has total utilization $U_{sum}(\tau) \approx 1.93 \leq 2$. As mentioned at the beginning of Section 1.5.2, τ is feasible on two processors. For any partitioning of this task set onto two processors, the total utilization of the tasks assigned to one of the processors is greater than one. Therefore, τ cannot be scheduled under PEDF on two processors so that all tasks meet their deadlines (or have bounded deadline tardiness).

More concretely, suppose that tasks T_1 and T_4 are assigned to processor 1 and tasks T_2 and T_3 are assigned to processor 2. An example schedule for τ on these two processors is shown in Figure 1.4(a). In this schedule, processor 1 is *overloaded* because the arriving jobs of T_1 and T_4 request 34 execution units every 24 time units while the processor can supply only 24 execution units. As a result, $T_{4,1}$ misses its deadline by two time units and $T_{1,3}$ misses its deadline by three time units. Overall, the maximum deadline tardiness for T_1 and T_4 is unbounded if their jobs arrive periodically. In contrast, if τ is scheduled on three processors, then all deadlines can be met, as shown in the example schedule in Figure 1.4(b). However, in this case, the overall capacity equivalent to approximately one processor remains unused in the long run.

1.5.3 Global Multiprocessor Scheduling

In contrast to partitioned scheduling, some global schedulers incur no utilization loss in implicit-deadline systems. Global algorithms can be further classified as either restricted or unrestricted. A scheduling algorithm is considered to be *restricted* if the scheduling priority of each job (for any given schedule) does not change once the job has been released. A scheduling algorithm is considered to be *unrestricted* if there exists a schedule in which some job changes its priority after it is released.

In this section, we discuss two restricted global scheduling algorithms, preemptive global EDF (Global EDF (GEDF)) and non-preemptive global EDF (Non-preemptive Global EDF (NPGEDF)); unrestricted algorithms are considered later in Section 2.1.2. Under both GEDF and NPGEDF, tasks are scheduled from a single priority queue on an EDF basis. The only difference between GEDF and NPGEDF is that jobs can be preempted under GEDF and cannot be preempted under NPGEDF.

Example 1.4. Consider the task system τ from Example 1.3. An example GEDF schedule for τ on two processors is shown in Figure 1.4(c). In this schedule, job $T_{4,1}$ misses its deadline at time 8 by one time unit. Note that, in this schedule, task T_4 migrates between processors 1 and 2. Figure 1.4(d) shows a NPGEDF schedule for τ. In this schedule, job $T_{4,1}$ meets its deadline. However, job $T_{1,2}$ misses its deadline by one time unit because it is blocked by lower-priority jobs of T_3 and T_4 during the time interval $[3, 5)$.

Similarly to PEDF, GEDF may leave up to half of the system's processing capacity unused if HRT schedulability is required. Particularly, an implicit-deadline sporadic task system τ with total utilization $U_{sum}(\tau)$ and $\max(u_i) \leq 1/2$ may need up to $\lceil 2 \cdot U_{sum}(\tau) - 1 \rceil$ processors in order to be HRT schedulable (Baruah, 2003). (Even more processors may be needed if $\max(u_i) > 1/2$.)

In contrast, for purely SRT systems, utilization loss can be eliminated. According to Devi and Anderson (2008b), for an implicit-deadline task system τ, bounded deadline tardiness is guaranteed under GEDF and NPGEDF if $U_{sum}(\tau) \leq m$. For the task system τ in Example 1.4, the maximum deadline tardiness is at most 8.5 under GEDF (see Section 2.1.1 for details).

We conclude this section by briefly mentioning one unrestricted global scheduler, namely, the PD^2 Pfair algorithm, which is one of the few optimal multiprocessor scheduling algorithms for implicit-deadline task systems. Any such task system τ, where tasks have integral execution times and periods and $U_{sum}(\tau) \leq m$, is HRT schedulable by PD^2 (Anderson and Srinivasan, 2004). Conversely, if $U_{sum}(\tau) > m$, then τ is infeasible. Unfortunately, the usage of PD^2 in practical settings may be limited due to its high preemption and migration overheads (Brandenburg et al., 2008a). PD^2 and other important unrestricted

schedulers are discussed in greater detail later in Section 2.1.2.

1.6   Limitations of the Sporadic Task Model

Modern embedded systems are becoming complex and distributed in nature. Such complexity may preclude efficient analysis using the periodic and sporadic models, thus making the schedulability results described in the prior sections inapplicable.

In this section, we use an example to illustrate some of the critical limitations of the sporadic task model that can arise during the analysis of a real application. We then briefly describe the streaming task model and the associated real-time calculus analysis framework, which circumvents these limitations and is widely used in the analysis of embedded systems.

Example 1.5. We consider an MPEG-2 video decoder application that has been studied previously in (Chakraborty et al., 2006; Phan et al., 2008). The originally-studied application, shown in Figure 1.5(a), is partitioned and mapped onto two processors. Processor 1 runs the VLD (variable-length decoding) and IQ (inverse quantization) tasks, while processor 2 runs the IDCT (inverse discrete cosine transform) and MC (motion compensation) tasks. The (coded) input bit stream enters this system and is stored in the input buffer B. The macroblocks (portions of frames of size 16×16 pixels) in B are first processed by task T_1 and the corresponding partially-decoded macroblocks are stored in the buffer B' before being processed by T_2. The resulting stream of fully decoded macroblocks is written into a playout buffer B'' prior to transmission by the output video device. In the above system, the coded input event stream arrives at a constant bit-rate.

Consider tasks T_1 and T_2, which are scheduled on separate processors. Suppose that jobs of T_1 arrive every $p_1 = 4$ time units, odd-indexed jobs require three execution units, and even-indexed jobs require one execution unit as shown in Figure 1.5(b). Such a situation is typical in MPEG decoding, where frames of different types have substantially different decoding times and come in repeating patterns. Suppose that jobs of T_2 are released in response to the completions of T_1's jobs and require three execution units each. Task T_1 can be described using the sporadic task model because its jobs are released p_1 time units apart and its worst-case execution time satisfies $e_1 \leq p_1$. However, T_2 cannot be described using the sporadic task model. As seen in Figure 1.5(b), the minimum inter-arrival time of T_2's jobs is two time units, while its worst-case execution time is three time units. Nevertheless, the response-time of T_2's jobs is bounded and is at most four time units. The sporadic task model also introduces pessimism when estimating long-term task execution requirements. The utilization of T_1 as defined by Definition 1.1
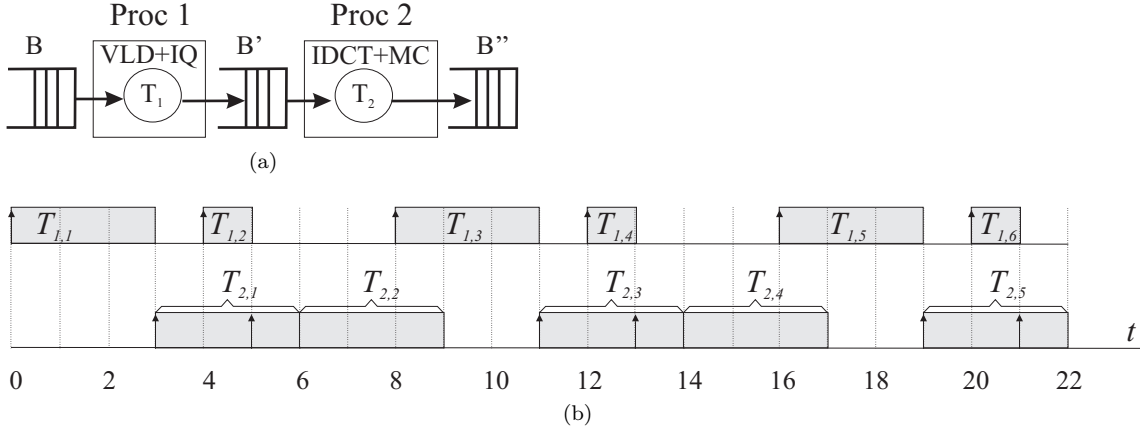
Figure 1.5: **(a)** MPEG Player application and **(b)** example schedule of tasks T_1 and T_2 in Example 1.5.

is $u_1 = 0.75$. However, task T_1's effective utilization is 0.5 as it consumes half of the capacity of one processor over sufficiently long time intervals.

In the example above, the sporadic task model is insufficient because it does not capture the long-term execution requirements of tasks or long-term job arrival patterns. The *multiframe* and *periodic with jitter* task models have been proposed to include these features in task descriptions (Mok and Chen, 1997). However, a more systematic approach to the analysis of communicating tasks such as those in Example 1.5 was enabled with the introduction of the *streaming task model* and the *real-time calculus* framework described next (Chakraborty et al., 2003, 2006).

1.7 Real-Time Calculus Overview

Real-time calculus is a specialization of *network calculus*, which was proposed by Cruz (1991a,b) and has been widely used to analyze communication networks. Real-time calculus specializes network calculus to the domain of real-time and embedded systems by, for example, adding techniques to model different schedulers and mode/state-based information (e.g., see (Phan et al., 2008)). A number of schedulability tests have also been derived based upon network calculus. We review some of these tests in Section 2.4.

In real-time calculus, jobs are invoked in response to external events. Timing properties of event streams are represented using upper and lower bounds on the number of events that can arrive over any time interval of a specified length. These bounds are given by functions $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$, which specify the maximum and minimum number of events, respectively, that can arrive at a processing/communication resource within any time interval of length Δ (or the maximum/minimum number of possible task activations within any Δ). The service offered by a resource is similarly specified us-
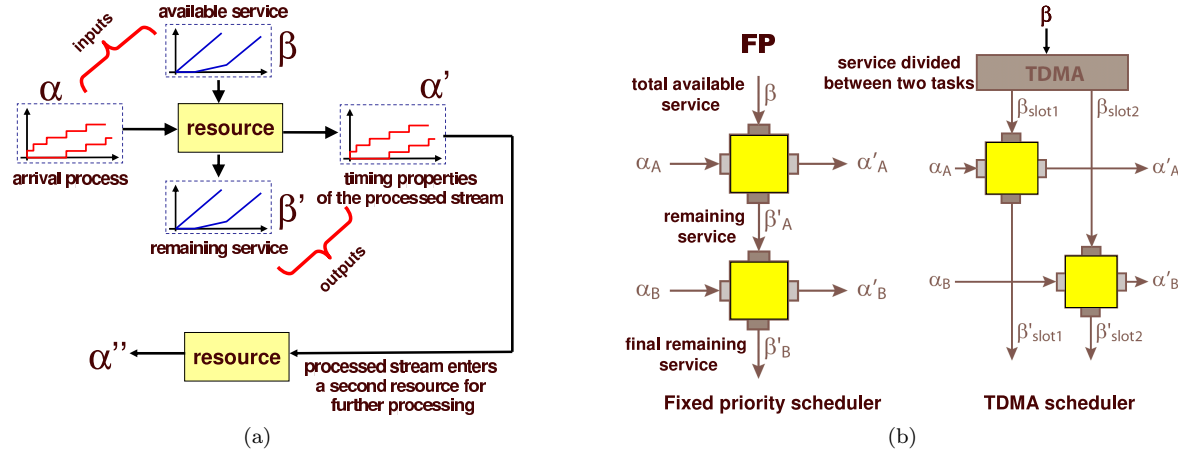
Figure 1.6: **(a)** Computing the timing properties of the processed stream using real-time calculus. **(b)** Scheduling networks for fixed priority and TDMA schedulers.

ing functions $\beta^u(\Delta)$ and $\beta^l(\Delta)$, which specify the maximum and minimum number of serviced events, respectively, within any interval of length Δ. Given the functions $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ corresponding to an event stream arriving at a resource, and the service $\beta^u(\Delta)$ and $\beta^l(\Delta)$ offered by it, it is possible to compute the timing properties of the processed stream and remaining processing capacity, i.e., functions $\alpha^{u\prime}(\Delta)$, $\alpha^{l\prime}(\Delta)$, $\beta^{u\prime}(\Delta)$, and $\beta^{l\prime}(\Delta)$, as illustrated in Figure 1.6(a), as well as the maximum backlog and delay experienced by the stream. As shown in the same figure, the computed functions $\alpha^{u\prime}(\Delta)$ and $\alpha^{l\prime}(\Delta)$ can then serve as inputs to the next resource on which this stream is further processed. By repeating this procedure until all resources in the system have been considered, timing properties of the fully-processed stream can be determined, as well as the end-to-end event delay and total backlog. This forms the basis for composing the analysis for individual resources, to derive timing/performance results for the full system.

Similarly, for any resource with tasks being scheduled according to some scheduling policy, it is also possible to compute service bounds ($\beta^u(\Delta)$ and $\beta^l(\Delta)$) available to its individual tasks. Figure 1.6(b) shows how this is done for the *fixed-priority* (FP) and *time-division-multiple-access* (TDMA) policies. As shown in this figure, for the FP policy, the *remaining* service after processing Stream A serves as the input (or, is available) to Stream B. On the other hand, for the TDMA policy, the total service $\beta(\Delta)$ is split between the services available to the two streams. Similar so called *scheduling networks* (Chakraborty et al., 2006) can be constructed for other scheduling policies as well. Various operations on the arrival and service curves $\alpha(\Delta)$ and $\beta(\Delta)$, as well as procedures for the analysis of scheduling networks on uniprocessors (and partitioned systems) have been implemented in the RTC (real-time calculus) toolbox (Wandeler and Thiele, 2006), which is a MATLAB-based library that can be used for modeling and
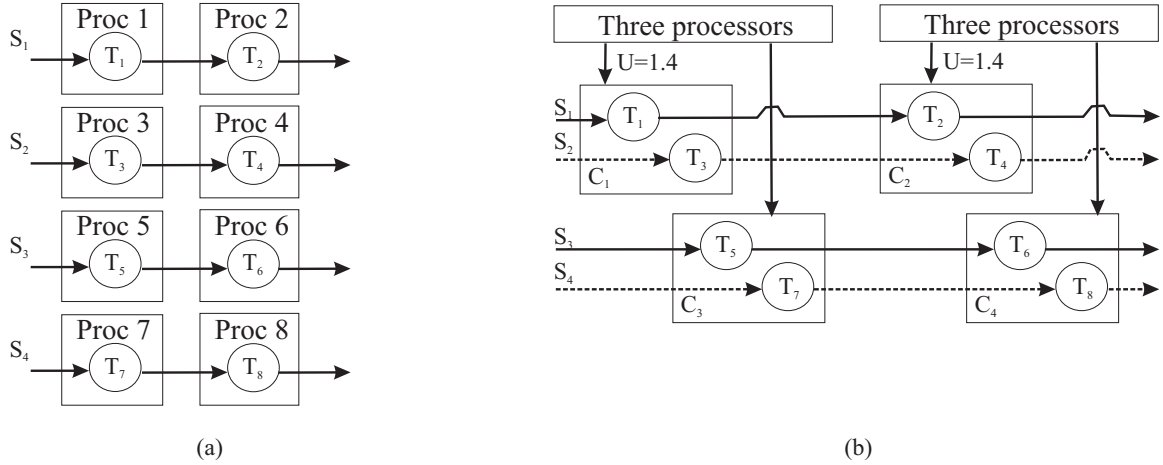
Figure 1.7: A complex multiprocessor multimedia application under **(a)** partitioning and **(b)** global scheduling.

analyzing distributed real-time systems.

1.8 Research Needed

With the needed background and concepts in place, we return to the subject of this dissertation, namely, extending compositional techniques for the design and analysis of real-time systems on multiprocessors. We motivate the open research questions in this area by looking at an example real-time multimedia system.

Consider an application consisting of four MPEG decoders similar to that in Figure 1.5(a) that process four video streams $S_1, \ldots, S_4$ as shown in Figure 1.7(a). Tasks T_1 and T_2 process stream S_1, tasks T_3 and T_4 process stream S_2, and so on. Suppose that each task requires 70% of the capacity of a dedicated processor. If partitioning is used, then the entire application requires eight processors to accommodate all tasks. However, since the cumulative utilization requirement is $0.7 \cdot 8 = 5.6$, six processors may be sufficient if global scheduling is used. Additionally, suppose that we want to isolate the tasks processing different groups of streams into containers as shown in Figure 1.7(b). Here, the tasks are encapsulated into four containers $C_1, \ldots, C_4$. Containers C_1 and C_3 are scheduled using the first three processors and containers C_2 and C_4 are scheduled using the remaining three processors. Such a setup would ensure isolation between two groups of streams if some tasks request more resources than provisioned.

Using the real-time application just described as motivation, we now formulate several problems that need to be solved in order for applications such as this to be analyzed and implemented successfully. We

17

first note that, since each task has a utilization of 70%, each container has a utilization of 140%, which requires the capacity of more than one processor. This poses the first problem.

(i) How can the processing capacity of a multiprocessor platform be allocated to a set of containers some of which require more than one processor? How can the supply available to each of the containers be characterized?

One potential solution, which is described and analyzed formally in Chapter 4, is to fully dedicate processor 1 and 40% of the capacity of processor 2 to running tasks in C_1. The remaining time on processors 2 and 3 can be used for running tasks in C_3. Containers C_2 and C_4 can be dealt with similarly.

Given a characterization of the supply for each container, tasks can be viewed as being scheduled on a set of partially-available processors. The problem of verifying timing constraints on a restricted-capacity platform has received some recent attention. However, these efforts only consider sporadic tasks with HRT constraints (Bini et al., 2009b; Anderson et al., 2006; Easwaran et al., 2009). We review this prior work in greater detail in Chapter 2. Allowing timing constraints to be soft for some tasks poses the second research problem.

(ii) Which scheduling algorithms can ensure SRT constraints (e.g., bounded tardiness or bounded maximum task response times) for workloads scheduled on a set of partially-available processors? Which such algorithms require the least processor supply?

Finally, we need to calculate the timing properties of the fully-processed streams, as well as the end-to-end event delay and total backlog. This poses the third research problem.

(iii) How can the properties mentioned above be analyzed for a set of streaming tasks scheduled using a global scheduler on a set of partially-available processors?

1.9 Thesis Statement

The main thesis of this dissertation, which attempts to answer the three research questions above, is the following.

With the exception of static-priority algorithms, virtually all previously studied global real-time scheduling algorithms ensure bounded deadline tardiness for implicit-deadline sporadic task systems. This property is preserved even if the processing capacity of some processors is not fully available, provided that the

long-term execution demand does not exceed the total available processing capacity. Well-studied global schedulers such as GEDF and First-In-First-Out (First-In-First-Out (FIFO)) ensure bounded maximum response times in systems with complex job arrival and execution patterns as described by the streaming task model. The use of such algorithms enables component-based systems with predominantly soft timing constraints to be built while incurring little or no utilization loss in settings where partitioning approaches are too costly in terms of needed processing resources.

1.10 Contributions

In this section, we briefly describe the contributions of this dissertation.

1.10.1 Generalized Tardiness Bounds

The first contribution we discuss is a generalized job prioritization rule originally proposed in (Leontyev and Anderson, 2008a, 2010) and tardiness bounds under it.

We found that the singular characteristic needed for tardiness to be bounded under a global scheduling algorithm is that a pending job's priority eventually (in bounded time) is higher than that of any future job. Global algorithms that do *not* have this characteristic (and for which tardiness can be unbounded) include static-priority algorithms such as the rate-monotonic (Rate-Monotonic (RM)) algorithm, and impractical dynamic-priority algorithms such as the earliest-deadline-*last* (Earliest Deadline Last (EDL)) algorithm, wherein jobs with earlier deadlines have *lower* priority. Global algorithms that *do* have this property include the EDF, FIFO, EDF-until-zero-laxity (Earliest Deadline Zero Laxity (EDZL)), and least-laxity-first (Least Laxity First (LLF)) algorithms. (EDZL is described later in Section 2.1.2 and LLF is described in Section 3.2.)

We establish a generalized tardiness result by considering a generic scheduling algorithm where job priorities are defined by points in time that may vary as time progresses. All of the algorithms mentioned above can be seen as special cases of this generic algorithm in which priorities are further constrained. Even the PD^2 Pfair algorithm (Anderson and Srinivasan, 2004), which uses a rather complex notion of priority, is a special case. In this dissertation, we present a derivation of a tardiness bound that applies to the generic algorithm if priorities are *window-constrained*: a job's priority at any time must correspond to a point in time lying within a certain time window. We also show that if this window constraint is violated, then tardiness can be unbounded. It is possible to define window-constrained prioritizations for EDF, FIFO, EDZL, LLF, and PD^2, as well as the earliest-pseudo-deadline-first (Earliest Pseudo-

Deadline First (EPDF)) Pfair algorithm, so these algorithms have bounded tardiness. (For EDF, EPDF, and PD^2, this was previously known.) For any other algorithm that may be devised in the future, our results enable tardiness bounds to be established by simply showing that prioritizations can be expressed in a window-constrained way (instead of laboriously devising a new proof).

The notion of a window-constrained priority is very general. For example, it is possible to describe hybrid scheduling policies by combining different prioritizations, *e.g.*, using a combination of EDF and FIFO in the same system. Priority rules can even change dynamically (subject to the window constraint). For example, if a task has missed too many deadlines, then its job priorities can be boosted for some time so that it receives special treatment. Or, if a single job is in danger of being tardy, then its prioritization may be changed so that it completes execution non-preemptively (provided certain restrictions hold — see Section 3.4.5). Tardiness also remains bounded if early-release behavior is allowed or if the capacity of each processor that is available to the (soft) real-time workload is restricted. In simplest terms, the main message is that, *for global scheduling algorithms, bounded tardiness is the common case, rather than the exception* (at least, ignoring clearly impractical algorithms such as EDL). For the widely-studied EDZL and LLF algorithms, and for several of the variants of existing algorithms just discussed, this dissertation is the first to show that tardiness is bounded. The proposed formulation of job priorities has been used by other researchers for the design and implementation of cache-aware multiprocessor real-time schedulers (Calandrino, 2009) and for devising new interrupt accounting techniques on multiprocessors (Brandenburg et al., 2009).

1.10.2   Processor Bandwidth Reservation Scheme

The second major contribution of this dissertation is a new multiprocessor scheduling approach for multi-level hierarchical containers that encapsulate sporadic SRT and HRT tasks. In this scheme, each container is allocated a specified *bandwidth*, which it uses to schedule its children (some of which may also be containers).

The bandwidth $w(H)$ is allocated to container H by means of reserving $\lfloor w(H) \rfloor$ processors for its children plus (if the bandwidth is not integral) the time occasionally available on an additional processor such that the total processor time supplied to H over a sufficiently long period of time Δ is approximately $w(H) \cdot \Delta$. The supply to the H's children is thus represented as a number of fully available processors plus at most one processor that is partially available. Given this allocation scheme, H's child tasks and containers are accommodated as follows. First, the set of child HRT tasks $\mathsf{HRT}(H)$ is encapsulated in a container C_{hrt} with an integral bandwidth $w(C_{hrt})$ so that $\mathsf{HRT}(H)$ is schedulable using PEDF.
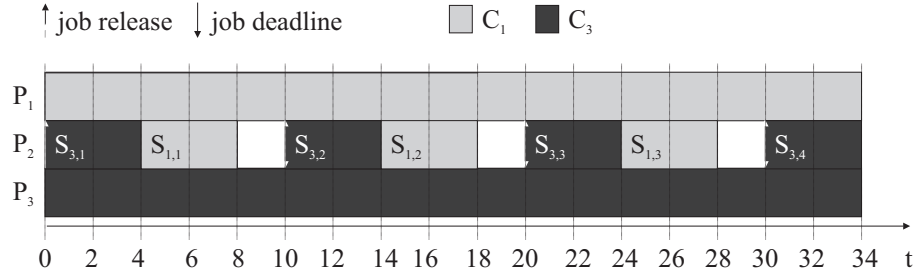
Figure 1.8: Illustration to container allocation in Example 1.6.

Second, each child container C_j is given $\lfloor w(C_j) \rfloor$ dedicated processors from the set of $\lfloor w(H) \rfloor$ processors dedicated to H. Third, for each child container C_j with a non-integral bandwidth, a *server task* $S_j(e_j, p_j)$ is created such that $u_j = w(C_j) - \lfloor w(C_j) \rfloor$. When task S_j is scheduled, tasks from C_j are scheduled. The set of SRT tasks and server tasks is scheduled together on processors that are not reserved for HRT tasks and child containers using an algorithm that ensures bounded tardiness for each task. Each child container C_j thus receives processing time approximately proportional to its requested bandwidth. Applying this strategy recursively, we can accommodate an entire container hierarchy.

Example 1.6. Consider the multimedia application introduced in Section 1.8. We define the bandwidth of container C_1 and C_3 to be $w(C_1) = w(C_3) = 1.4$. Since the bandwidth is non-integral, for each of the containers C_1 and C_3 we dedicate $\lfloor w(C_1) \rfloor = \lfloor w(C_3) \rfloor = 1$ processor and construct periodic server tasks $S_1(4, 10)$ and $S_3(4, 10)$ with utilizations $u_1 = u_3 = 0.4$. Figure 1.8 shows an example schedule in which processors 1 and 3 are dedicated to containers C_1 and C_3 and the server tasks are scheduled using EDF on processor 2. Each container thus receives the capacity of approximately 1.4 processors over sufficiently long time intervals.

Our scheme is novel in that, in a system with only SRT tasks, no utilization loss is incurred (assuming that system overheads are negligible—such overheads will cause some loss in any scheme in practice). This statement is true, provided the goal is to schedule SRT tasks so that their tardiness is bounded, no matter how great the bound may be. The scheduling scheme we present also allows HRT tasks to be supported. However, such support may incur some utilization loss. These tradeoffs are discussed in detail in Section 4.5.

In addition to presenting our overall scheme, we also present the results of experiments conducted to assess its usefulness. In these experiments, our scheme exhibited performance—in terms of both necessary processing capacity and tardiness—comparable to that of schemes that exhibit good performance but are oblivious to containers (and hence, do not provide any container isolation).

1.10.3 Multiprocessor Extensions to Real-Time Calculus

The third major contribution is a framework for the analysis of multiprocessor processing elements with streaming tasks where *the constituent processors are managed according to a global multiprocessor scheduling algorithm.* Such processing elements can be used for building complex applications that cannot be analyzed using state-of-the-art multiprocessor scheduling techniques and that must be over-provisioned, wasting processing resources, if analyzed using conventional real-time calculus. Sporadic and streaming task sets under GEDF, and static-priority schedulers, can be analyzed in this framework.

Our work is different from prior efforts that assume implicit deadlines, full processor availability, and non-zero tardiness for each task (Devi, 2006; Devi and Anderson, 2005, 2006). In one recent paper, Bini et al. (2009a) presented a HRT schedulability test for GEDF for systems where processors can be partially available. However their work also assumes that tasks are sporadic and have constrained deadlines. In contrast, the task model and the scheduler assumed in our proposed framework are very general.

The core of our framework is a procedure for checking that arbitrary pre-defined job response times $\{\Theta_1, \ldots, \Theta_n\}$ are not violated under a restricted global scheduling algorithm on a platform with a minimum cumulative capacity $\mathcal{B}(\Delta)$. Note that, if relative deadlines and tardiness thresholds are specified for tasks, then checking pre-defined job response times is equivalent to checking whether a job completes within its relative deadline plus its tardiness threshold.

In settings where response-time bounds $\{\Theta_1, \ldots, \Theta_n\}$ are not known, they must be determined. In addition to giving a test that checks pre-defined response-time bounds, we propose closed-form expressions for calculating response-time bounds directly from task and supply parameters for a family GEDF-like schedulers such as GEDF and FIFO. The obtained expressions for response-time bounds are similar to those for calculating tardiness bounds under GEDF proposed by Devi and Anderson (2008b). It is also possible to refine the obtained response-time bounds by incrementally decreasing them and running the aforementioned test procedure to see if the smaller bounds are also valid.

Once maximum job response-time bounds $\{\Theta_1, \ldots, \Theta_n\}$ are determined, we use them to characterize the sequences of job completion events for each task T_i in terms of arrival functions $\alpha_i^{u\prime}(\Delta)$ and $\alpha_i^{l\prime}(\Delta)$, and the remaining cumulative processor supply $\mathcal{B}'(\Delta)$ (see Figure 1.9). The calculated stream and supply outputs can serve as inputs to subsequent processing elements, thereby resulting in a compositional technique.
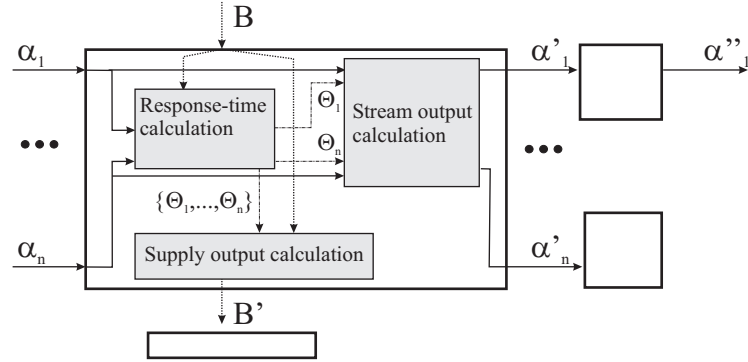
Figure 1.9: A multiprocessor element analyzed using multiprocessor real-time calculus.

1.11   Summary

In this chapter, we have motivated the research in this dissertation with the need to support component-based systems on multiprocessor platforms. We have presented the widely-studied sporadic task model and some important multiprocessor scheduling algorithms for it. We have also shown that this model may be insufficient for describing workloads in component-based systems. After stating several open research questions pertaining the design and analysis of multiprocessor component-based systems, we gave a list of contributions of this dissertation addressing these questions.

The rest of the dissertation is organized as follows. In Chapter 2, we review prior work on multiprocessor soft real-time and hierarchical scheduling. In Chapter 3, we present our generalized tardiness bound proof. In Chapter 4, we present our hierarchical scheduling framework. In Chapter 5, we present multiprocessor extensions to real-time calculus. In Chapter 6, we summarize the work presented in this dissertation and outline directions for future work.

Chapter 2

Prior Work

In this chapter, we review prior work that is relevant to the focus of this dissertation on multiprocessor schedulability analysis, hierarchical scheduling, and real-time calculus. In Section 2.1, we present some schedulability results for implicit-deadline task systems under GEDF and NPGEDF and illustrate other important multiprocessor scheduling algorithms. In Section 2.2, we review two recent approaches for checking the schedulability of constrained-deadline task systems on fully-available multiprocessor platforms by Baruah (2007) and Bertogna et al. (2008). In this dissertation, we extend the techniques proposed by Baruah by incorporating more expressive task and processor supply models. However, we adopt some ideas from Bertogna et al. as well. In Section 2.3, we present three multiprocessor hierarchical scheduling frameworks for HRT and SRT tasks. One of these frameworks, proposed by Bini et al. (2009a), is based upon the test by Bertogna et al. Another one, proposed by Easwaran et al. (2009), is based upon Baruah's test. In addition to presenting the ideas behind these frameworks, we also compare them to the hierarchical scheduling scheme proposed in this dissertation. Finally, in Section 2.4, we review prior work on real-time calculus.

2.1 Multiprocessor Scheduling

In this section, we discuss some results concerning HRT and SRT schedulability of implicit-deadline sporadic task systems under GEDF, and present several unrestricted global multiprocessor schedulers.

2.1.1 GEDF Schedulability Results

One way to ensure task timing constraints in a SRT system is to treat all deadlines as hard (i.e., set $\Theta_i = 0$). Perhaps partly because of that, most prior work on GEDF has focused on hard real-time

schedulability tests (Baker, 2003; Baruah, 2007; Baruah and Baker, 2008; Bertogna et al., 2008). If such a test passes, then each task is guaranteed zero tardiness. Unfortunately, ensuring zero tardiness under GEDF may severely restrict system utilization. According to Goossens et al. (2003), an implicit-deadline task system τ can be guaranteed to meet all deadlines on m processors under GEDF if

$$m \geq \left\lceil \frac{U_{sum}(\tau) - 1}{1 - \max(u_i)} \right\rceil.$$

The task set from Example 1.3 in Section 1.5.3 may thus require $m \geq \left\lceil \frac{U_{sum}(\tau) - 1}{1 - \max(u_i)} \right\rceil = \left\lceil \frac{1.93 - 1}{1 - 3/4} \right\rceil = \lceil 3.72 \rceil = 4$ processors in order to meet all job deadlines. Because the total utilization is $U_{sum}(\tau) \approx 1.93$, half of the platform's processing capacity will be unused in this case.

As mentioned earlier in Section 1.5.3, Devi and Anderson (2008b) showed that, for an implicit-deadline task system τ, bounded deadline tardiness is guaranteed under GEDF and NPGEDF if $U_{sum}(\tau) \leq m$. That is, for SRT systems, utilization loss can be eliminated. Let

$$\lambda = \begin{cases} U_{sum} - 1 & \text{if } U_{sum} \text{ is integral,} \\ \lfloor U_{sum} \rfloor & \text{otherwise.} \end{cases}$$

Then deadline tardiness for task T_i under GEDF is at most

$$e_i + \frac{E_L - \min(e_i)}{m - U_L}, \tag{2.1}$$

where E_L is the sum of λ largest task WCETs and U_L is the sum of $\lambda - 1$ largest task utilizations. Similar expression were obtained in (Devi and Anderson, 2008b) for NPGEDF and for the case when tasks consist of interleaving preemptive and non-preemptive regions. If tardiness thresholds Θ_i are specified, then we can calculate tardiness bounds Θ_i' using (2.1) and then verify that $\Theta_i' \leq \Theta_i$ holds for each task T_i. Unfortunately, this method cannot be applied if some tardiness thresholds are small because $\Theta_i' \geq e_i$ for each $T_i \in \tau$. This precludes the analysis of systems with mixed HRT and SRT constraints. Also, (2.1) is applicable only to implicit-deadline task systems.

For constrained- and arbitrary-deadline task systems, several HRT schedulability tests for GEDF have been proposed (Baker, 2003; Baruah, 2007; Baruah and Baker, 2008; Bertogna et al., 2008). All of these tests assume full processor availability and incur some utilization loss in order to guarantee hard deadlines. Unfortunately, no research has been done yet concerning the calculation or verification of tardiness bounds in constrained- and arbitrary-deadline task systems scheduled on multiprocessors.

One can argue that in order to verify pre-defined tardiness thresholds, task T_i's relative deadline can be set to $D_i + \Theta_i$, where D_i is the old relative deadline and Θ_i is T_i's allowed tardiness threshold, and then the HRT schedulability of the modified system can be verified. Though this method is valid for the verification of timing constraints, it changes the relative priority of jobs of different tasks, which may be unacceptable.

Additionally, introducing tardiness thresholds allows a job's timing constraint to be decoupled from its scheduling priority. For example, an arbitrary-deadline task system τ with relative deadlines $\{D_1, \ldots, D_n\}$ that is not HRT schedulable under GEDF can be SRT schedulable for a different set of relative deadlines $\{D'_1, \ldots, D'_n\}$ and tardiness thresholds $\{\Theta'_1, \ldots, \Theta'_n\}$ such that $D'_i + \Theta'_i = D_i$ for each i. The idea of decoupling priorities and timing constraints is elaborated on in greater detail in Chapter 3.

2.1.2 Unrestricted Global Multiprocessor Scheduling

Unrestricted schedulers allow job priorities to change at runtime. In this section, we briefly present the earliest-deadline-zero-laxity (EDZL), earliest-pseudo-deadline-first (EPDF), Pfair PD^2, and least-local-remaining-execution-first (Least Local Remaining Execution First (LLREF)) algorithms.

EDZL algorithm. EDZL, which was first proposed by Cho et al. (2002), is a conventional GEDF algorithm with an added "safety rule." Under EDZL, a job is prioritized by its deadline unless it is in danger of missing its deadline. This moment is detected by calculating the job's laxity, which is the difference between the current time and the latest time when the job can be scheduled so that it meets its deadline. Jobs with zero laxity are given the highest priority.

Example 2.1. Consider the task set from Example 1.4. An example EDZL schedule for it is shown in Figure 2.1(a). In this schedule, at time 0, all jobs have positive laxity (i.e., if scheduled immediately, each job will complete before its deadline). Therefore, jobs $T_{1,1}$ and $T_{2,1}$, which have the smallest absolute deadlines are scheduled. At time 2, job $T_{4,1}$ has zero laxity (i.e., if scheduled later, then it will miss its deadline). By the zero laxity rule, its priority is raised and $T_{4,1}$ executes uninterruptedly until its deadline.

In the literature, several schedulability tests have been proposed for EDZL (Cirinei and Baker, 2007; Piao et al., 2006; Wei et al., 2007). It has been shown that EDZL can schedule any task set that is schedulable under GEDF (Cho et al., 2002). However, for any U such that $U \leq m$ and $U > m \cdot (1 - 1/e)$, where e is Euler's number, there exists at least one implicit-deadline task system τ such that $U_{sum}(\tau) = U$
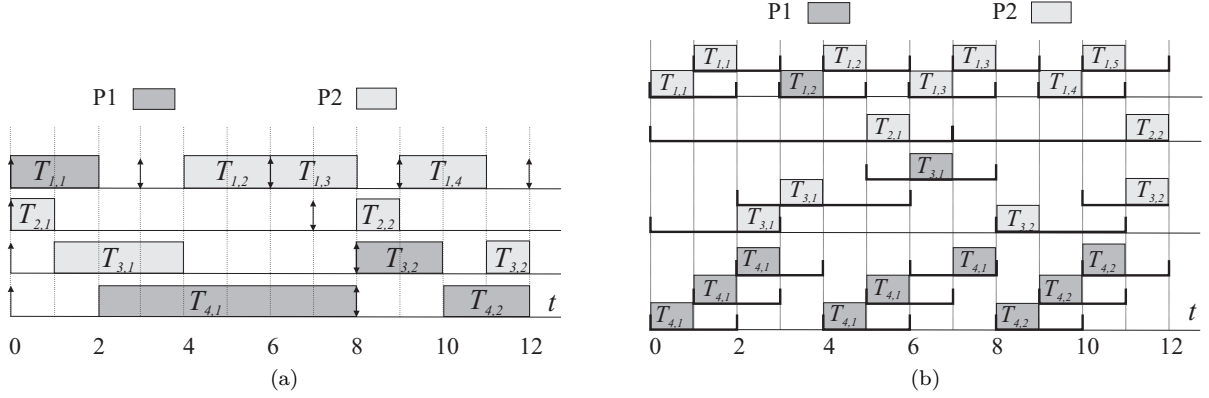
Figure 2.1: Example **(a)** EDZL and **(b)** EPDF schedules.

that is unschedulable by EDZL (Wei et al., 2007).

EPDF algorithm. Under the EPDF Pfair algorithm (Devi and Anderson, 2008a), task periods and execution times are assumed to be integral, and each task T_i is represented by a sequence of unit-length schedulable entities called *subtasks*, denoted T_i^j, where $j \geq 1$. Each subtask T_i^j has two attributes associated with it, a *release time* r_i^j and a *deadline* d_i^j. The interval $[r_i^j, d_i^j)$ is called the *window* of T_i^j. Subtask T_i^j becomes available for execution at time r_i^j and has higher priority than subtask T_x^y if $d_i^j < d_x^y$. Deadline ties are resolved arbitrarily but consistently.

Example 2.2. In considering EPDF scheduling examples, we assume (for simplicity) that jobs are released in a synchronous periodic fashion, in which case $r_i^j = \lfloor \frac{i-1}{u_i} \rfloor$ and $d_i^j = \lceil \frac{i}{u_i} \rceil$ (see (Anderson and Srinivasan, 2004)). Figure 2.1(b) shows an EPDF schedule for the task set τ from Example 1.3. In this schedule, each subtask executes within its respective window, which is shown in bold. Thus, all tasks meet their deadlines.

Allowing early releases can reduce job response times as the following example illustrates.

Example 2.3. Figure 2.2 shows two EPDF schedules of a task $T_1(3, 8)$. Inset (a) shows a schedule in which early releases are not allowed. In this schedule, each subtask executes within its respective window. The time between $T_{1,1}$'s release and completion is six time units. A schedule in which early releases are allowed is shown in Figure 2.2(b). In this schedule, each subtask executes immediately after its predecessor completes. In this schedule, the response time of $T_{1,1}$ is three time units.

It has been shown that EPDF correctly schedules any implicit-deadline task system τ (with integral execution times and periods) on m processors if $U_{sum}(\tau) \leq \frac{3 \cdot m + 1}{4}$ (Devi and Anderson, 2008a).

27
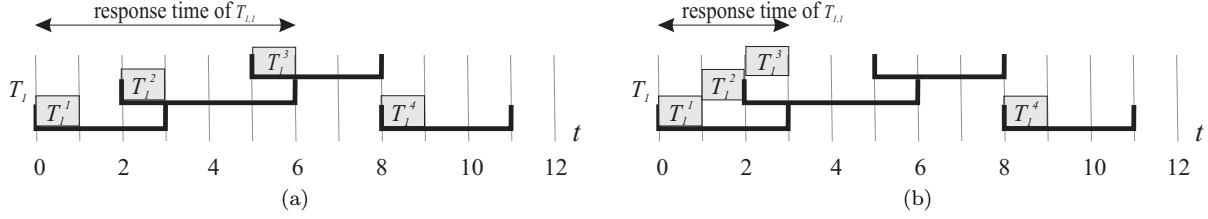
Figure 2.2: EPDF schedules from Example 2.3 **(a)** without and **(b)** with early releases.

Additionally, EPDF ensures a maximum tardiness bound of q quanta if $\max_{T_i \in \tau}(u_i) \leq \frac{q+2}{q+3}$ and $U_{sum}(\tau) \leq m$ (Devi and Anderson, 2009).

PD^2 and LLREF algorithms. PD^2 differs from EPDF in that two special tie-breaking rules are used in the event of a deadline tie. As mentioned earlier in Section 1.5.3, PD^2 is one of the few optimal multiprocessor scheduling algorithms for implicit-deadline task systems. The LLREF scheduling algorithm, which was proposed by Cho et al. (2006), is another example of an optimal multiprocessor scheduler. Unfortunately, it is optimal only for periodic workloads as it requires that the arrival time of every job be known *a priori*.

2.2 Multiprocessor Schedulability Tests

Most schedulability tests for global algorithms are based on a simple principle proposed by Baker (2003). First, let job $T_{\ell,q}$ be the first job (in some ordering) to miss its deadline. Second, calculate the minimum amount of competing demand due to jobs of other tasks that is necessary for $T_{\ell,q}$ to miss its deadline. This gives a necessary condition for a deadline violation. Finally, calculate an upper bound on the competing demand. Setting the lower bound to be greater than the upper bound gives a sufficient condition for schedulability. Different tests, however, may have different time complexities, and may also differ in predictive power, depending on the assumptions made when calculating the upper and lower bounds on the competing demand.

2.2.1 SB-Test

The schedulability test and analysis techniques proposed by Baruah (2007) are important because their introduction initiated a collection of new results about the schedulability of sporadic task sets (including arbitrary-deadline task sets) under GEDF and several other algorithms on multiprocessor platforms. We henceforth refer to this schedulability test as the "SB-test."

28

The test considers a constrained-deadline ($D_i \leq p_i$) task system τ scheduled on m identical fully-available processors. The test is derived by considering an interval $[r_{\ell,q} - A_\ell, r_{\ell,q} + D_\ell]$, where $T_{\ell,q}$ is the problem job that misses its deadline, A_ℓ is a parameter with range $[0, A_\ell^{\max}]$, and $A_\ell^{\max}$ is a constant that depends on the parameters of the tasks in τ, m, and the index ℓ. The length of the interval of interest is thus $A_\ell + D_\ell$. During this interval, the demand due to competing equal-or-higher-priority jobs that can interfere with $T_{\ell,q}$ is considered. Then the following three steps are performed:

S1: The minimum execution demand due to tasks other than T_ℓ and jobs of T_ℓ that have higher priority than $T_{\ell,q}$ that is necessary for $T_{\ell,q}$ to miss its deadline is computed. This demand is $m \cdot (A_\ell + D_\ell - e_\ell)$.

S2: An upper-bound on the competing demand $M^*(A_\ell)$, which depends on τ, m, and A_ℓ, is calculated.

S3: The upper bound $M^*(A_\ell)$ is compared with the lower bound $m \cdot (A_\ell + D_\ell - e_\ell)$. If, for each task $T_k \in \tau$, $M^*(A_k) \leq m \cdot (A_k + D_k - e_k)$ holds for all $A_k \in [0, A_k^{\max}]$, then no job misses its deadline.

Example 2.4. Consider task system τ in Example 1.4 in Section 1.5.3. It has been shown that τ is not HRT schedulable using GEDF on $m = 2$ processors. In this example, we show that SB-test will fail for τ. Consider a schedule in Figure 2.3 in which the problem job $T_{\ell,q} = T_{4,1}$ misses its deadline by ϵ time units. Additionally, suppose that the execution time of job $T_{1,2}$ is ε time units and the execution time of job $T_{3,1}$ is $2 + \varepsilon$ time units. We next set $A_\ell = 0$ and consider the problem interval $[r_{\ell,q}, r_{\ell,q} + D_\ell] = [0, 8]$. For this interval, an upper-bound on competing demand $M^*(A_\ell)$ is at least the competing demand due higher-priority jobs $T_{1,1}$, $T_{1,2}$, $T_{2,1}$, and $T_{3,1}$, which is 2, ε, 1, and $1 + \varepsilon$, respectively. Note that even though the execution time of job $T_{3,1}$ is $2 + \epsilon$, this job and the problem job $T_{4,1}$ execute in parallel during the interval $[2, 3)$ so the competing demand due to job $T_{3,1}$ is smaller. (In Figure 2.3, the competing demand is shown with black.) The total competing demand is thus $4 + 2\varepsilon$. We thus have $M^*(A_\ell) \geq 4 + 2 \cdot \varepsilon > m \cdot (A_\ell + D_\ell - e_\ell) = 2 \cdot (0 + 8 - 6) = 4$, and hence, the SB-test will fail for τ.

Theorem 2.1. (Proved in (Baruah, 2007).) *The time complexity of the SB-test is pseudo-polynomial if there exists a constant c such that $U_{sum}(\tau) \leq c < m$.*

After its introduction, the SB-test was extended in several ways. First, a test with somewhat lower time complexity for the analysis of hard real-time arbitrary-deadline sporadic task systems under GEDF
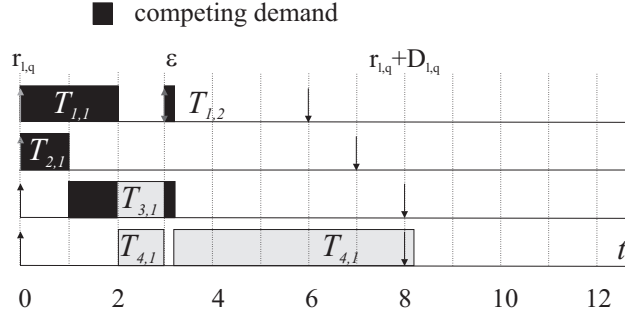
Figure 2.3: An application of SB-test in Example 2.4.

was proposed (Baruah and Baker, 2008). Second, Guan et al. (2008, 2009) used the proposed techniques to derive new schedulability tests for arbitrary-deadline task systems under NPGEDF and static-priority scheduling. Third, Leontyev and Anderson (2008b) independently derived schedulability tests for arbitrary-deadline task systems with *specified* tardiness constraints under GEDF and NPGEDF. (These tests were developed independently from those by Guan et al.) The distinguishing property of the test proposed by Leontyev and Anderson is that task sets with *mixed* HRT and SRT constraints can be analyzed. We later discovered that restricted-capacity systems and more general task models can also be analyzed using the approach of the original SB-test. This, however, required significant modifications to the original analysis as described in detail in Chapter 5.

2.2.2 BCL-Test

As mentioned earlier, the SB-test has pseudo-polynomial time complexity. However, if a task set fails the test, it is not clear how "bad" is it, i.e., by how much deadlines can be missed. Also, the unmodified SB-test is only applicable to fully preemptive GEDF. Bertogna et al. (2008) attempted to address these issues by proposing a framework consisting of a family of schedulability tests that are applicable not only to GEDF but also to fixed-priority scheduling and any general work-conserving scheduling algorithm. We refer to this framework and its derivatives as the "BCL-test." Similarly to the SB-test, the BCL-test assumes constrained deadlines and full processor availability. Additionally, all time quantities are assumed to be integral. The theorem below establishes a schedulability condition for GEDF.

Theorem 2.2. (Proved in (Bertogna et al., 2008).) *A task set τ is schedulable under GEDF on m processors if, for each task $T_k \in \tau$,*

$$\sum_{i \neq k} \min(J_{i,k}, D_k - e_k + 1) < m \cdot (D_k - e_k + 1), \tag{2.2}$$

where

$$J_{i,k} = \left\lfloor \frac{D_k}{p_i} \right\rfloor \cdot e_i + \min(e_i, D_i - \left\lfloor \frac{D_k}{p_i} \right\rfloor \cdot p_i). \tag{2.3}$$

The schedulability test in Theorem 2.2 has time complexity of $O(n^2)$, where $n = |\tau|$. Bertogna et al. also proposed a more accurate iterative version of the test, which has pseudo-polynomial time complexity of $O(n^2 \cdot \max(D_i))$.

Example 2.5. Consider the task system τ in Example 1.4 in Section 1.5.3. By Theorem 2.2 it is schedulable on $m = 3$ processors. Consider, for example, task $T_k = T_1$. For this task, the right-hand side of (2.2) is $m \cdot (D_1 - e_1 + 1) = 3 \cdot (3 - 2 + 1) = 6$. We now calculate the left-hand side of (2.2). By (2.3), $J_{2,1} = \left\lfloor \frac{D_1}{p_2} \right\rfloor \cdot e_2 + \min(e_2, D_2 - \left\lfloor \frac{D_1}{p_2} \right\rfloor \cdot p_2) = \left\lfloor \frac{3}{7} \right\rfloor \cdot 1 + \min(1, 7 - \left\lfloor \frac{3}{7} \right\rfloor \cdot 7) = 1$. $J_{3,1} = 3$ and $J_{4,1} = 6$ are calculated similarly. Thus, if $k = 1$, then the left-hand side of (2.2) is $\min(J_{2,1}, D_1 - e_1 + 1) + \min(J_{3,1}, D_1 - e_1 + 1) + \min(J_{4,1}, D_1 - e_1 + 1) = \min(1, 2) + \min(3, 2) + \min(6, 2) = 5$, and hence (2.2) holds holds for $k = 1$. The other tasks can be tested similarly.

Experiments presented by Bertogna et al. (2008) showed that the BCL-test has greater accuracy than previously-developed tests for GEDF and fixed-priority scheduling. However, the BCL- and SB-tests do not dominate each other. That is, there exist task sets deemed schedulable by the BCL-test for which the SB-test fails and visa versa.

2.3 Multiprocessor Hierarchical Scheduling

The schedulability tests for fully-available platforms can be modified to enable the analysis of restricted-capacity platforms. This need arises during the design and analysis of virtually any hierarchically scheduled system in which the processing capacity of a multiprocessor has to be shared among different components. Depending on how the processing capacity available to a component is restricted, and component tasks are scheduled within the available capacity, different hierarchical scheduling frameworks can be constructed. In this section, we describe three multiprocessor hierarchical scheduling frameworks developed recently. These frameworks will be illustrated using an example component-based system below.

Example 2.6. Let C_1, C_2, and C_3 be three components encapsulating implicit-deadline sporadic tasks as shown in Figure 2.4. The total utilization of tasks within each component is 7/6, 4/3, and 3/2,
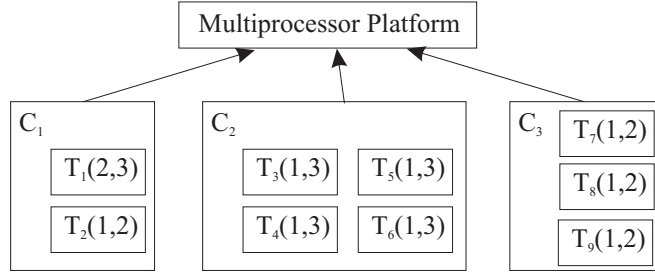
Figure 2.4: A component-based system in Example 2.6.

respectively. Thus, each of the components would require the capacity of more than one processor when scheduled on a multiprocessor platform. If each component is given two processors, then six processors are needed in order for all encapsulated tasks to meet their deadlines (or have bounded tardiness).

2.3.1 Megatask Scheduling

The first hierarchical scheduling framework we discuss is *megatask* scheduling, which was originally proposed by Anderson et al. (2006) in the context of cache-aware Pfair real-time scheduling. It can also be straightforwardly used for component scheduling. In this framework, implicit-deadline tasks, for which parallel co-scheduling needs to be discouraged are grouped into *megatasks* which become schedulable entities. Each megatask γ_j is characterized by rational *weight* $W_j \geq \sum_{T_k \in \gamma_j} u_k$, which represents the long-term processor share requested by the megatask. We let $I_j = \lfloor W_j \rfloor$ be the integral part of γ_j's weight and $f_j = W_j - I_j$ be the fractional part.

The proposed megatask scheduling scheme is a two-level hierarchical approach. The root-level scheduler is PD^2, which schedules all megatasks and tasks that do not belong to any megatask (free tasks). Pfair scheduling with megatasks is a straightforward extension to ordinary Pfair scheduling. For each megatask γ_j, I_j processors are statically assigned to this megatask and a dummy or fictitious, synchronous, periodic task F_j of weight f_j is created. The remaining $m - \sum_{\gamma_j} I_j$ processors are allocated at runtime to the fictitious tasks and free tasks by the root-level PD^2 scheduler. Whenever task F_j is scheduled, an additional processor is allocated to γ_j. Within the time available to a megatask, a second-level PD^2 scheduler is used for the encapsulated tasks. However, second-level tasks may miss their deadlines due to limited processor availability. Anderson et al. derived tardiness bounds for second-level tasks and proposed to inflate megatask weights if the deadlines of the second-level tasks should be met.

Example 2.7. In the system from Example 2.6, we represent the components C_1, C_2, and C_3 by megatasks γ_1, γ_2, and γ_3 with weights $W_1 = 7/6$, $W_2 = 4/3$, and $W_3 = 3/2$. A schedule representing
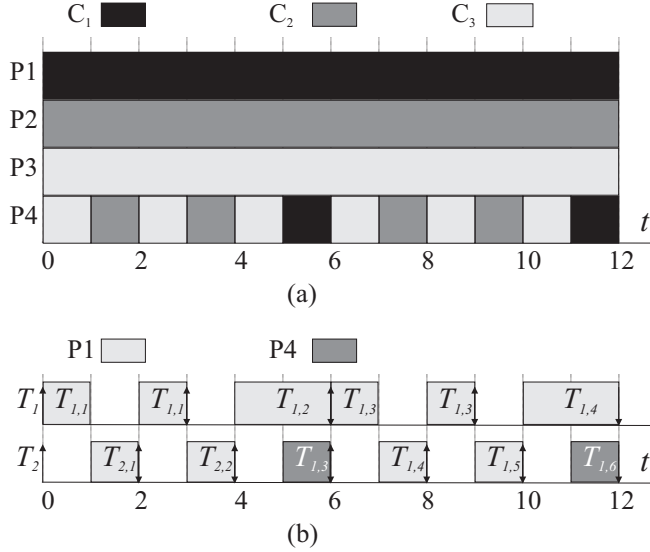
Figure 2.5: **(a)** Allocation of processor time to components (megatasks) and **(b)** a schedule of the tasks encapsulated in C_1 in Example 2.7.

processor time allocated to the three components is shown in Figure 2.5(a). In this schedule, each of the first three processors is exclusively dedicated to the respective component and the fourth processor is shared among the three components in accordance with the fractional parts of the megatask weights $(1/6, 1/3,$ and $1/2,$ respectively). As seen, in the long run, component C_1 is given seven units of processor allocation every six time units. Figure 2.5(b) shows a PD^2 schedule of tasks T_1 and T_2 encapsulated in C_1. These tasks are scheduled using the time available to C_1. Note that all deadlines are met in this schedule. It can be shown that all tasks in this example system meet their deadlines when scheduled using megatasks. (Deadline misses are likely to occur in systems with per-task utilizations at least 0.5.)

Megatask scheduling is very similar to the hierarchical scheme proposed in this dissertation in that the execution requirement of each component is described by a single value that upper-bounds the long-term utilization of constituent tasks. Also, the two schemes share the idea of minimizing execution parallelism by allocating time using some integral number of processors plus at most one additional processor, which is allocated at a certain rate. The megatask scheme is, however, limited only to two-level task hierarchies, implicit-deadline tasks, and Pfair scheduling. The quantum-based nature of Pfair scheduling also can incur significant scheduling overhead (Brandenburg et al., 2008a), which motivates research on alternative techniques for component scheduling, one of which is presented in the next section.

33

2.3.2 Virtual Cluster Scheduling

As opposed to megatask scheduling, the virtual cluster (VC) scheduling framework proposed by Easwaran et al. (2009) can use various scheduling policies for allocating processor time to components and for scheduling tasks within components. The name of the framework comes from the way component processor supply is specified. Each component is allocated time using a *cluster* of processors, some of which may be partially available (hence the term *virtual*). This is a generalization of physical cluster scheduling wherein components are scheduled on non-intersecting sets of physical processors (Calandrino et al., 2007; Chuprat and Baruah, 2008). The supply required by a component is characterized as follows.

Definition 2.1. (Easwaran et al., 2009) (Multiprocessor periodic resource model (MPR)) A multiprocessor periodic resource model $\mu =< \Pi, \Theta, m' >$, where $\Theta \leq m' \cdot \Pi$, specifies that an identical multiprocessor platform collectively provides Θ units of execution in every Π time units with concurrency at most m'; at any time instant, at most m' physical processors are allocated in this resource model. Θ/Π denotes the resource bandwidth of model μ.

Example 2.8. Consider again Example 2.6. For components C_1, C_2, and C_3, we specify resource models as $\mu_1 =< 6, 7, 2 >$, $\mu_2 =< 3, 4, 2 >$, and $\mu_3 =< 2, 3, 2 >$, respectively. (These parameters are also shown in Figure 2.6(a).) For example, component C_1 is supplied at least seven execution units every six time units on at most two processors.

Given the execution requirements of individual first-level components as in the MPR model, Easwaran et al. proposed the following method of allocating time on physical processors. For each component C_i with resource model $\mu_i =< \Pi_i, \Theta_i, m_i >$, create a set of m_i implicit-deadline periodic server tasks $\{T_1^{[i]}, \ldots, T_{m_i}^{[i]}\}$ with the following parameters.

$$T_1^{[i]} = T_2^{[i]} = \cdots = T_{m_i-1}^{[i]} = (\Pi_i, \Pi_i) \qquad\qquad T_{m_i}^{[i]} = (\Theta_i - (m_i - 1) \cdot \Pi_i, \Pi_i) \qquad (2.4)$$

The server tasks from all first-level components are scheduled together on m physical processors. Whenever task $T_j^{[i]}$ is scheduled on a processor, a task that belongs to C_i is scheduled on that processor using an internal scheduling policy. If server tasks of component C_i meet their deadlines, then C_i is supplied processor time according to its model parameters. In this case, C_i is allocated time on $m_i - 1$ fully available processors (via server tasks $T_1^{[i]}, \ldots, T_{m_i-1}^{[i]}$) plus the time on an additional processor allocated (via server task $T_{m_i}^{[i]}$) at a rate equal to the fractional part of C_i's requested bandwidth. This allocation scheme is similar to that in megatask scheduling except that various schedulers can be used

	Π_i	Θ_i	m_i	
C_1	6	7	2	$T_1^{[1]}$ (6,6) $T_2^{[1]}$ (1,6)
C_2	3	4	2	$T_1^{[2]}$ (3,3) $T_2^{[2]}$ (1,3)
C_3	2	3	2	$T_1^{[3]}$ (2,2) $T_2^{[3]}$ (1,2)

(a)

(b)

(c)
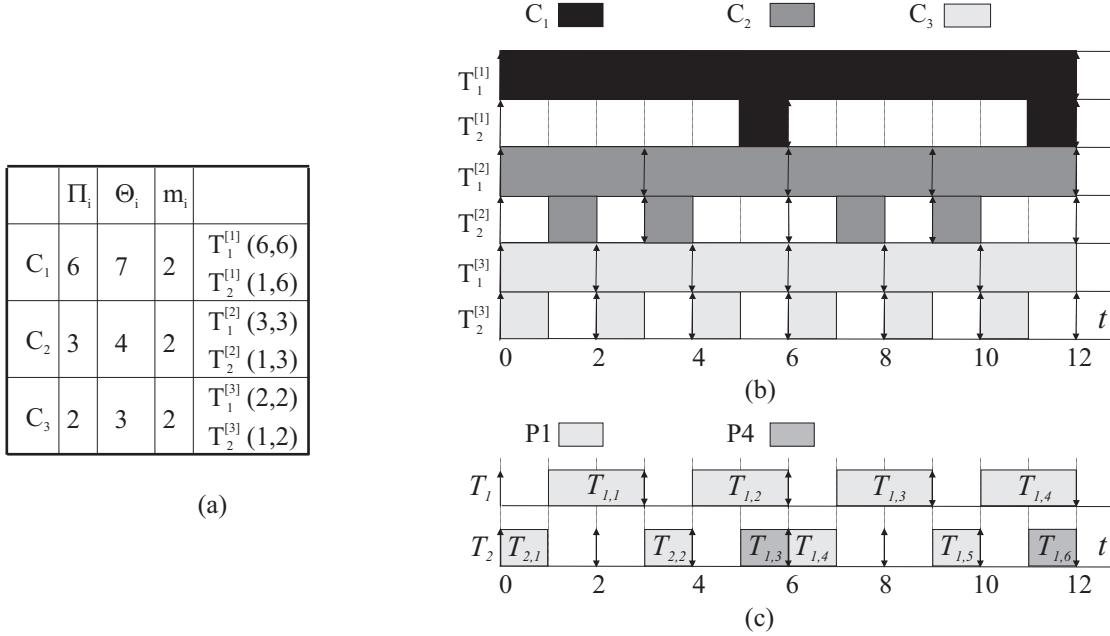
Figure 2.6: Virtual cluster scheduling in Examples 2.8 and 2.9.

for server tasks as long as their deadlines are met. To check the schedulability of component tasks under GEDF, Easwaran et al. proposed a modification of the SB-test that incorporates restricted supply.

Example 2.9. In Example 2.8, for components C_1, C_2, and C_3, we construct server tasks using (2.4). Their parameters are shown in inset (a) of Figure 2.6. Inset (b) of Figure 2.6 shows a PD^2 schedule of the server tasks on four processors (so that their deadlines are met). As seen, component C_1 receives 7 execution units every 6 time units involving at most 2 processors. Inset (c) of Figure 2.6 shows a GEDF schedule of tasks T_1 and T_2 encapsulated in C_1. In this schedule, all deadlines are met.

The VC framework provides several scheduling algorithms and analysis for checking the HRT schedulability of constrained-deadline tasks organized into components. Thus, it can be seen as an alternative to the hierarchical scheduling scheme proposed in this dissertation. However, there is significant distinction between the two frameworks. The current analysis of the VC framework supports only HRT constrained-deadline tasks. The need to satisfy hard real-time constraints incurs some utilization loss. Though the authors provide a procedure that can generate a resource model with minimum bandwidth for a component based upon the parameters of its constituent tasks, the use of GEDF for intra-component scheduling makes utilization loss unavoidable. This loss is further exacerbated if components are nested within each other, though Easwaran et al. do not investigate this issue in their paper. In contrast, the hierarchical scheduling framework proposed in this dissertation is focused on ensuring SRT constraints

(bounded tardiness) so that GEDF can be used as the intra-component scheduler with no utilization loss even for arbitrarily deep container hierarchies. (HRT tasks are also supported in our scheme though some utilization can be lost if such tasks are present.) The two frameworks thus meet different needs.

2.3.3 Parallel-Supply Function Abstraction

The supply model proposed for virtual cluster scheduling conceals some information that may be useful for analysis as illustrated in an example below.

Example 2.10. (Bini et al., 2009a) Suppose that the processor time on two identical unit-speed processors is supplied to component C as shown in Figure 2.7(c). In this schedule, both processors are not available during the interval $[0, 2)$, processor 1 is available during intervals $[2, 4)$ and $[6, 8)$, and processor 2 is available during the interval $[4, 8)$. The availability pattern then repeats every eight time units. When this supply is described in terms of the MPR resource model, it has budget $\Theta = 8$, period $\Pi = 8$, and maximum parallelism $m' = 2$. Hence, this formalism loses the potentially useful information that some processor is available for 6 units of time out of every 8 and two processors are available simultaneously for 2 time units.

To expose more information about the restricted-capacity platform, the parallel-supply function abstraction has been proposed by Bini et al. (2009a).

Definition 2.2. Let $Y_j(\Delta)$ be the minimum guaranteed amount of time available on at most $j \geq 1$ processors over any interval of length Δ. The *parallel-supply function* for a platform consisting of m processors is defined by the set $\{Y_1(\Delta), \ldots, Y_m(\Delta)\}$.

Example 2.11. Figure 2.7(a) shows the available processor time for container C_1 in Example 2.9. Figure 2.7(b) shows the parallel-supply function $\{Y_1(\Delta), Y_2(\Delta)\}$ for this container. During any time interval one processor is always available, and hence, $Y_1(\Delta) = \Delta$. During the interval $[0, 5)$, only one processor is available, and hence, $Y_1(\Delta) = Y_2(\Delta)$ for all $\Delta \leq 5$. During the interval $[5, 6)$, two processors are available, and hence, $Y_2(\Delta)$ grows faster than $Y_1(\Delta)$ for $\Delta \in [5, 6)$. As seen, the function $Y_1(\Delta)$ captures the fact that one processor is always available to C_1, and the existence of an additional available processor is captured by $Y_2(\Delta)$. In general, $Y_m(\Delta)$ describes the cumulative processor time available on the entire platform over any interval of length Δ.

Example 2.12. Now consider the supply from Example 2.10. Its parallel-supply function is shown in Figure 2.7(d). As seen, the minimum guaranteed amount of time available during any interval of length
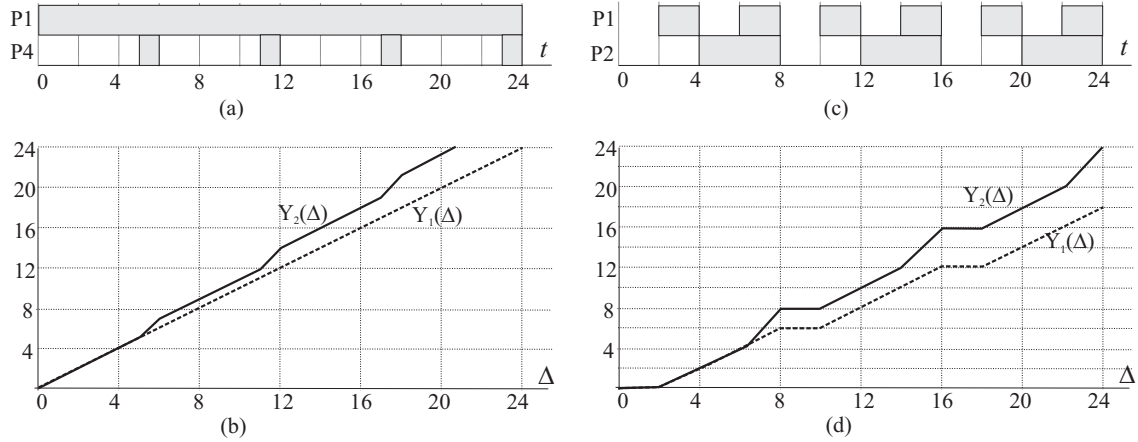
Figure 2.7: Parallel-Supply Function abstraction in Examples 2.10–2.11.

at most 2 is zero, because both processors are not available during the intervals $[0, 2)$ and $[8, 10)$, which are of length 2. During the intervals $[2, 8)$ and $[10, 16)$, one of the processors is available at each time, and hence, $Y_1(\Delta) = \Delta - 2$ for $\Delta \leq 8$. within this interval. Two processors are available during the interval $[6, 8)$ and hence, within this interval, $Y_2(\Delta)$ grows faster than $Y_1(\Delta)$.

Using the newly-developed parallel-supply function abstraction, Bini et al. proposed two novel HRT schedulability tests for constrained-deadline task systems scheduled using GEDF on a restricted-capacity platform. Their work is relevant to the multiprocessor real-time calculus extensions described in Chapter 5 in the following aspects. In Chapter 5, we study streaming task systems scheduled using a global algorithm on a platform whose capacity is restricted. However, we use more general job arrival and execution models and assume that the supply is described using *only* a cumulative supply function (i.e., $Y_m(\Delta)$).

2.4 Schedulability Analysis using Real-Time Calculus

As mentioned earlier in Section 1.7, the streaming task model and real-time calculus framework circumvent some critical limitations of the sporadic task model and allow the analysis of component-based systems to be performed in a more systematic way. In the real-time calculus framework, a component-based system is decomposed into a collection of simple processing elements (PEs). For a PE, the timing characteristics of its output event streams and remaining supply can be computed from the input arrival and supply functions. The calculated outputs serve as inputs for subsequent PEs (see Figure 1.6(a)). In this section, we briefly describe how the outputs can be calculated for FP and EDF scheduling.

Below, we will use the operators $\otimes$ and $\oslash$, called *min-plus convolution* and *min-plus deconvolution*,

respectively, that extend the plus and minus operators on functions in the min-plus algebra (LeBoudec and Thiran, 2001). The operators $\overline{\otimes}$ and $\overline{\oslash}$, called *max-plus convolution* and *max-plus deconvolution*, respectively, play a similar role in the max-plus algebra. Using these operators in real-time calculus allows the expressions for output streams to be written in a concise manner.

$$(f \otimes g)(\Delta) = \inf_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\}$$

$$(f \oslash g)(\Delta) = \sup_{\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\}$$

$$(f \,\overline{\otimes}\, g)(\Delta) = \sup_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\}$$

$$(f \,\overline{\oslash}\, g)(\Delta) = \inf_{0\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\}$$

Greedy processing component. A streaming task T with arrival curves $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ that is exclusively scheduled on a partially-available processor with supply β^u and β^l as shown in Figure 2.8(a) is one of the basic blocks in the real-time calculus framework. This configuration is called *a Greedy Processing Component* (GPC). Assuming that β^u and β^l are given in terms of serviced jobs per time unit, the output events for T and remaining supply functions for GPC can be calculated as follows (Wandeler, 2006).

$$\alpha^{u\prime}(\Delta) = \min\{((\alpha^u \otimes \beta^u) \oslash \beta^l)(\Delta), \beta^u(\Delta)\}$$

$$\alpha^{l\prime}(\Delta) = \min\{((\alpha^l \oslash \beta^u) \otimes \beta^l)(\Delta), \beta^l(\Delta)\}$$

$$\beta^{u\prime}(\Delta) = ((\beta^u - \alpha^l) \,\overline{\oslash}\, 0)(\Delta)$$

$$\beta^{l\prime}(\Delta) = ((\beta^l - \alpha^u) \,\overline{\otimes}\, 0)(\Delta)$$

A fixed-priority processing element is constructed by connecting several GPCs hierarchically as shown in Figure 2.8(b).

EDF element. The analysis of an EDF-based processing element is slightly more tricky. In the EDF case, each streaming task T_i is additionally characterized by the relative deadline D_i and worst- and best-case job execution times $e_i^{\max}$ and $e_i^{\min}$, respectively. Additionally, under EDF, the supply functions $\beta^u(\Delta)$ and $\beta^l(\Delta)$ are specified as available processor time over any interval of length Δ.

It should be noted that, if components with different supply representation are connected together, then a conversion should be done. The details of conversions between event-based and time-based
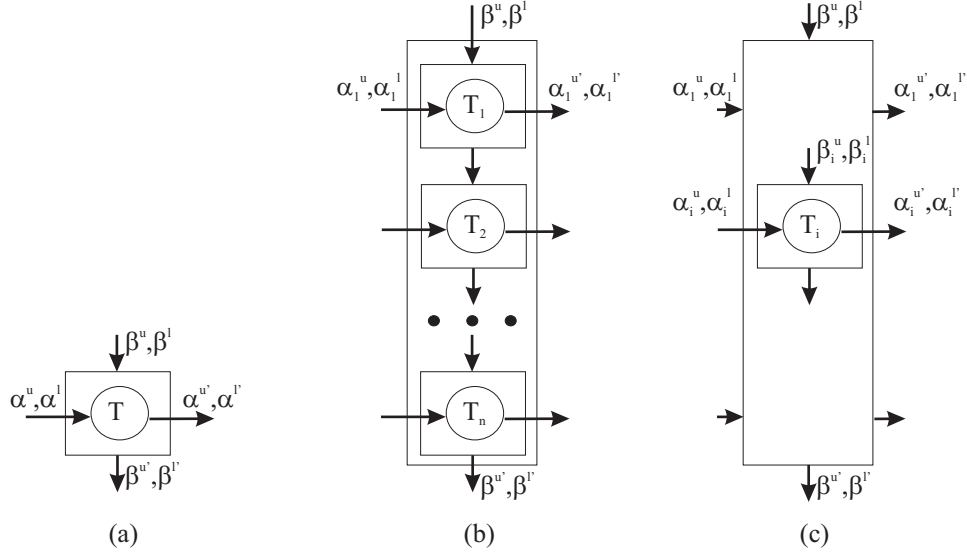
Figure 2.8: **(a)** Greedy, **(b)** FP, and **(c)** EDF processing elements in real-time calculus framework.

representations of supply can be found in Section 4.1 of (Wandeler, 2006).

The set of streaming tasks τ is schedulable under uniprocessor EDF if (2.5) below holds for each $\Delta \geq 0$ (Wandeler and Thiele, 2006).

$$\sum_{T_i \in \tau} \alpha_i^u(\Delta - D_i) \cdot e_i^{\max} \leq \beta^l(\Delta) \tag{2.5}$$

In (2.5), the left-hand side is the maximum processor time needed by jobs with release times and deadlines within any interval of length Δ to complete before their deadlines. The output supply curves for EDF processing are be calculated similarly to those for the GPC (Wandeler and Thiele, 2006).

$$\beta^{u\prime}(\Delta) = ((\beta^u - \sum_{T_i \in \tau} e_i^{\min} \cdot \alpha_i^l) \; \overline{\oslash} \; 0 \;)(\Delta)$$

$$\beta^{l\prime}(\Delta) = ((\beta^l - \sum_{T_i \in \tau} e_i^{\max} \cdot \alpha_i^u) \; \overline{\otimes} \; 0 \;)(\Delta)$$

The calculation of the output event streams is more involved. Each task T_i is treated as being executed on a GPC with supply functions $\beta_i^u(\Delta)$ and $\beta_i^l(\Delta)$ defined below.

$$\beta_i^u(\Delta) = \beta^u(\Delta)/e_i^{\min}$$

$$\beta_i^l(\Delta) = \frac{((\beta^l - \sum_{T_j \neq T_i} e_j^{\max} \cdot \alpha_j^u) \; \overline{\otimes} \; 0)(\Delta)}{e_i^{\max}}$$

Let $\alpha_i^{u''}(\Delta)$ and $\alpha_i^{l''}(\Delta)$ be the output event functions calculated for T_i scheduled using a GPC with supply $\beta_i^u(\Delta)$ and $\beta_i^l(\Delta)$. The output functions for T_i under EDF scheduling are

$$\alpha_i^{u'}(\Delta) = \min\{\alpha_i^u(\Delta + D_i), \lceil \alpha_i^{u''}(\Delta) \rceil\}$$
$$\alpha_i^{l'}(\Delta) = \max\{\alpha_i^l(\Delta - D_i), \lfloor \alpha_i^{l''}(\Delta) \rfloor\}.$$

Example 2.13. In this example, we illustrate basic real-time calculus analysis using a variant of the embedded automotive system from prior work (Wandeler, 2006). Figure 2.9 shows an integrated radio/navigation system running on processors CPU1 and CPU2 connected with a communication bus BUS1. CPU1 runs graphical user interface and computational navigation tasks. CPU2 runs software that receives music over the radio and monitors the traffic message channel (TMC). Task T_1 is invoked when the user wants to change the sound volume. There could be at most 32 such requests per second (see the input labeled α_{vol} in the figure). After the user's request is processed by task T_1, a four-byte change-volume message is transmitted to task T_3 by communication task C_1 on BUS1. Task T_3 changes the sound volume and sends a four-byte message back to task T_2 that updates the screen showing the volume change.

Similarly, task T_4 receives TMC messages (typically 300 messages per 15 minutes) with traffic information over the radio (input α_{tmc}). Task T_4 performs initial decoding of these messages by extracting feature and location information and passes this information using 64-byte messages to task T_5. Task T_5 finishes the decoding, maps the features using location database, and displays relevant changes on the screen.

We have analyzed the system using basic real-time calculus analysis. We have assumed strictly periodic input event streams and system parameters as summarized in Table 2.1. We have found that the delay between user input and screen update is at most 18ms and the delay between the receipt of a TMC message and screen update is at most 218ms.

In recent papers, real-time calculus has been extended in several directions. First, some concepts of timed automata have been incorporated into real-time calculus to improve the accuracy of the analysis (Huang et al., 2007; Phan et al., 2008). Second, the basic analysis was extended to allow cyclic dataflow graphs (Thiele and Stoimenov, 2009). Third, the properties of some power-saving algorithms have been investigated using real-time calculus techniques (Chen et al., 2009). Fourth, real-time interfaces were introduced and procedures were proposed that allow *assumptions* on input streams and supply to be calculated from *assumptions* on processed streams (Chakraborty et al., 2006). Real-time
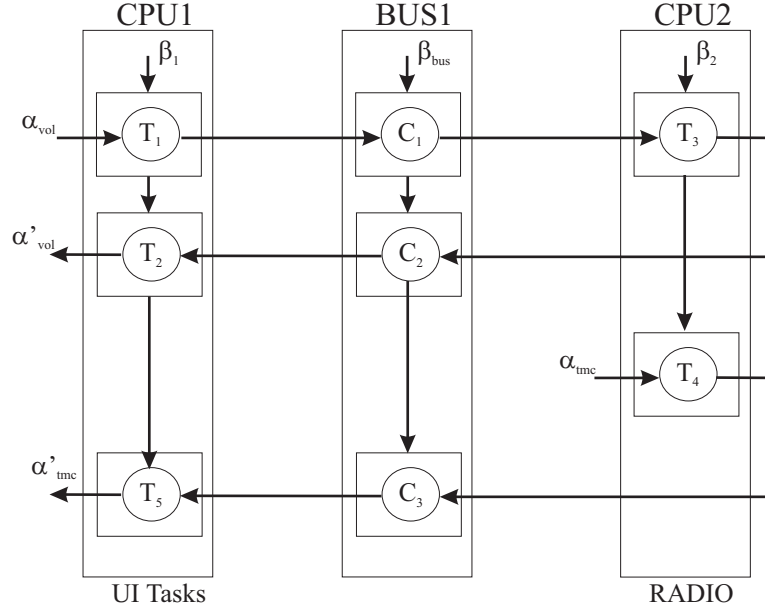
Figure 2.9: Scheduling network of an embedded automotive application in Example 2.13.

Table 2.1: System parameters in Example 2.13.

Number of instructions in T_1	10^5
$—//— T_2$	5×10^5
$—//— T_3$	10^5
$—//— T_4$	10^6
$—//— T_5$	5×10^6
Message size of C_1	4 bytes
$—//— C_2$	4 bytes
$—//— C_3$	64 bytes
CPU1 speed	100 MIPS
CPU2 speed	11 MIPS
BUS1 speed	72 kbps

interfaces enable easy checks of *compatibility* for any pair of connected components.

The ideas of describing the task workload and available processor time using general demand and supply functions has been used for the analysis of network packet and task scheduling. In (Sariowan et al., 1995; Cruz, 1995), the authors examine various scheduling policies (including EDF) that provide quality-of-service guarantees to network packets transferred over a communication link. In (Wu et al., 2005), the authors derive schedulability conditions for various task models under uniprocessor static-priority scheduling. Our research differs from these prior efforts in that we consider task systems scheduled on a *multiprocessor* using a *global* scheduler. We also made our framework compatible to real-time calculus so that new analysis could be easily integrated into existing software tools.

2.5   Summary

In this chapter, we have presented schedulability results for sporadic task systems scheduled under GEDF on a multiprocessor platform with full processor availability. Some of these results were later adopted for checking schedulability if processor availability is limited. The analysis of restricted-capacity platforms is crucial for building frameworks for hierarchical scheduling. Three of such frameworks have been presented in this chapter. We concluded this chapter by presenting real-time calculus analysis for uniprocessor FP and EDF schedulers.

Chapter 3

Generalized Tardiness Bounds

In this chapter[1], we present generalized tardiness bounds for implicit-deadline task systems scheduled on a multiprocessor.

This chapter is organized as follows. In Sections 3.1 and 3.2, we present some additional model assumptions and our scheduling framework. Then, in Section 3.3, we present our tardiness-bound derivation. In Section 3.4, we discuss some special cases and possible extensions to the analysis. As discussed later, tardiness may be different under different scheduling algorithms. In Section 3.5, we present results from experiments conducted to assess such differences. Section 3.6 concludes the chapter.

3.1 Preliminaries

We consider the problem of scheduling a set of implicit-deadline SRT tasks $\tau = \{T_1, \ldots, T_n\}$ as defined in Section 1.3 on an m-processor platform as defined in Section 1.4. All time quantities considered in this chapter are assumed to be real numbers. In addition to $U_{sum}(\tau) \leq m$ (see Section 1.5.2), we assume

$$U_{sum}(\tau) \leq \sum_{k=1}^{m} \widehat{u_k}, \tag{3.1}$$

where $\widehat{u_k}$ is the long-term utilization available on processor k (see Definition 1.2 in Section 1.4). Otherwise, tardiness may grow unboundedly. In this chapter, we henseforth omit the parameter τ in U_{sum}.

We assume that eligible jobs are placed into a single global ready queue. When choosing a new job to schedule, the scheduler selects (and dequeues) the ready job of highest priority. As reiterated in

---

[1]Contents of this chapter previously appeared in the following paper:
Leontyev, H. and Anderson, J. (2009a). Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Systems*. To appear.

Definition 3.3 in Section 3.3, a job is *ready* if it is eligible and its predecessor (if any) has completed execution. Job priorities are determined as follows.

Definition 3.1. (prioritization functions) Associated with each job $T_{i,j}$ is a function of time $\chi(T_{i,j}, t)$ defined for $t \geq 0$ and called its *prioritization function*. If $\chi(T_{i,j}, t) < \chi(T_{k,h}, t)$, then the priority of $T_{i,j}$ is higher than the priority of $T_{k,h}$ at time t. We assume that, when comparing priorities, any ties are broken arbitrarily but consistently. That is, if, $\chi(T_{i,j}, t) = \chi(T_{k,h}, t)$ and $\chi(T_{i,j}, t') = \chi(T_{k,h}, t')$, where $t \neq t'$, then the tie is broken in favor of $T_{i,j}$ at time t iff it is broken in favor of $T_{i,j}$ at time t'.

3.2   Example Mappings

We now show how to describe several well-known scheduling policies in our framework, using the two-processor task set $\tau = \{T_1(1, 3), T_2(2, 3), T_3(1, 4), T_4(3, 4)\}$ executing on two fully-available processors as an example. Unless stated otherwise, we assume $e_{i,j} = e_i$ and $\epsilon_{i,j} = r_{i,j}$ in these examples, for each job $T_{i,j}$.

Example 3.1. Figure 3.1(a) shows a schedule for τ under the global EDF algorithm. In this case, since jobs are prioritized by deadline, it suffices to define $\chi(T_{i,j}, t) = d_{i,j}$ for each $T_{i,j}$. In Figure 3.1(a), the value of $\chi(T_{i,j}, t)$ is shown for each job $T_{i,j}$ using a black circle labeled $\chi_{i,j}$.

Example 3.2. Figure 3.1(b) shows a schedule for τ under the global RM algorithm. In this case, $T_{i,j}$ should have priority over $T_{k,h}$ if $i < k$ (since the tasks in τ are ordered by increasing periods). Thus, we can simply define $\chi(T_{i,j}, t) = i$ for each job $T_{i,j}$, as shown.

Example 3.3. Figure 3.1(c) shows a schedule for τ under the global FIFO algorithm (which, by definition, schedules jobs non-preemptively). In this case (assuming no early releases), it suffices to define $\chi_{i,j}(t) = r_{i,j}$ for each job $T_{i,j}$, as shown. (Note that, if early releases are allowed, then this prioritization may not reflect the actual job arrival order.)

Example 3.4. Interestingly, the definition of $\chi(T_{i,j}, t)$ is flexible enough to allow *combinations* of scheduling policies to be specified. For example, we can prioritize the jobs of $T_1, \ldots, T_3$ on an EDF basis and those of T_4 on a FIFO basis by defining $\chi(T_{i,j}, t) = d_{i,j}$ for $1 \leq i \leq 3$, and $\chi(T_{4,j}, t) = r_{4,j}$. A schedule for this hybrid policy is shown in Figure 3.1(d). It is also possible to mix RM and EDF prioritizations (even though such a scheme would not have window-constrained priorities). For example, if task T_1 needs to be statically prioritized over all other tasks, then we can set $\chi(T_{1,j}, t) = -1$ for all jobs of T_1 and $\chi(T_{i,j}, t) = d_{i,j}$ for all jobs of other tasks.
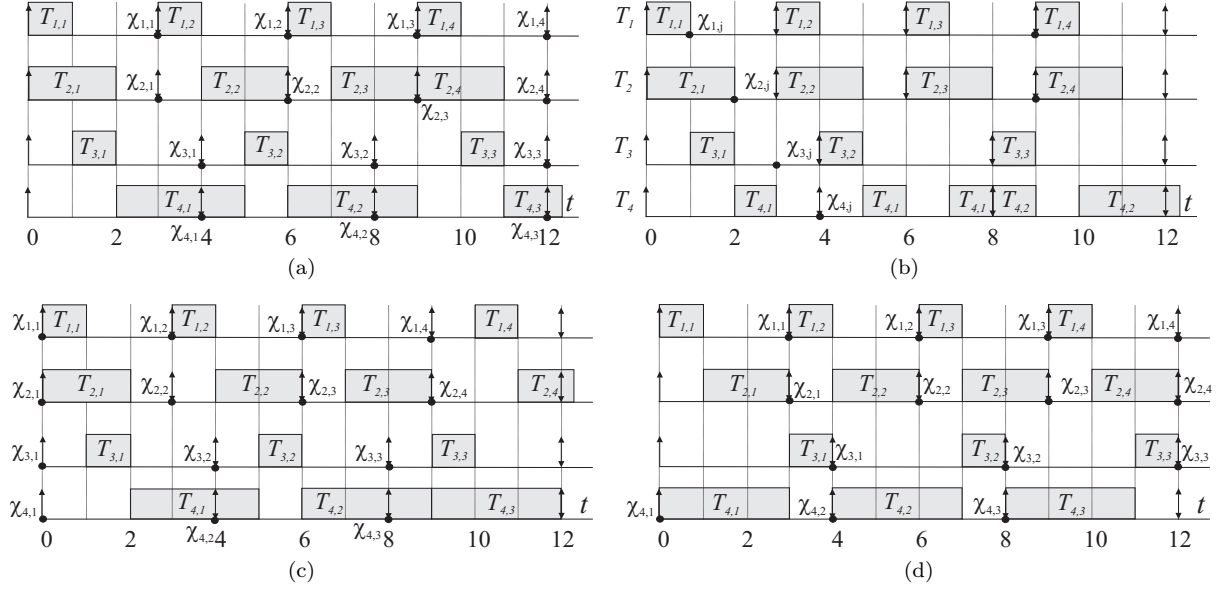
Figure 3.1: **(a)** Example 3.1 (GEDF). **(b)** Example 3.2 (global RM). **(c)** Example 3.3 (global FIFO). **(d)** Example 3.4 (hybrid global scheduler).

Example 3.5. So far we have considered only fixed job-priority algorithms, wherein the priority $\chi(T_{i,j}, t)$ is constant during job $T_{i,j}$'s execution. We now consider a slightly more complicated example, namely the global LLF scheduling algorithm (Liu, 2000). The *laxity* or *slack* of a job $T_{i,j}$ at time t is defined as

$$slack_{i,j}(t) = d_{i,j} - t - (e_i - \delta_{i,j}(t)), \tag{3.2}$$

where $\delta_{i,j}(t)$ is the amount of time for which $T_{i,j}$ has executed before t. If a job does not miss its deadline, then its slack is always non-negative; if it does miss its deadline, then its slack becomes negative at some time prior to its deadline. According to LLF, $T_{i,j}$ has higher priority than $T_{k,h}$ at time t if $slack_{i,j}(t) < slack_{k,h}(t)$. To capture this, we can simply define $\chi(T_{i,j}, t) = d_{i,j} - (e_i - \delta_{i,j}(t))$ for each job $T_{i,j}$. Because this definition depends on $\delta_{i,j}(t)$, $\chi(T_{i,j}, t)$ is not constant, as in the prior examples, but is time-dependent. Assuming that it is updated only at integral points in time, $\chi(T_{i,j}, t+1) := \chi(T_{i,j}, t) + 1$, if $T_{i,j}$ executes during the interval $[t, t+1)$, and $\chi(T_{i,j}, t+1) := \chi(T_{i,j}, t)$, otherwise.

Figure 3.2 shows an LLF schedule for τ where ties are broken in favor of jobs currently executing. Because χ-values change with time, they are not shown in the schedule, as earlier, but are depicted separately in Table 3.1. The table shows the value of $\chi(T_{i,j}, t)$ for the earliest pending job $T_{i,j}$ of each task T_i where $0 \leq t \leq 11$.

Example 3.6. The EDZL algorithm introduced in Example 2.1 in Section 2.1.2, can be specified as well.

45

Table 3.1: χ-values in Example 3.5.

Time t	$\chi(T_{1,j},t)$	$\chi(T_{2,j},t)$	$\chi(T_{3,j},t)$	$\chi(T_{4,j},t)$
0	2	1	3	1
1	2	2	3	2
2	2	–	3	3
3	5	4	3	–
4	5	5	7	5
5	5	–	7	6
6	8	7	7	7
7	8	8	7	–
8	8	–	11	9
9	11	10	11	10
10	11	11	11	11
11	11	–	11	–
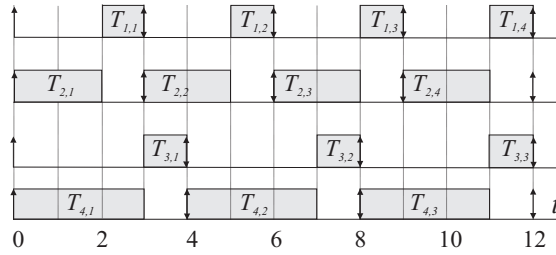
Figure 3.2: Example 3.5 (global preemptive LLF).

In this case, $\chi(T_{i,j},t)$ is set to $d_{i,j}$ (as in EDF) when $T_{i,j}$ is released, and is reset to $d_{i,j}-(e_i-\delta_{i,j}(t)) \leq d_{i,j}$ (as in LLF) when $T_{i,j}$'s slack becomes zero, where $\delta_{i,j}(t)$ is as defined earlier. To our knowledge, EDZL has not been considered previously in systems where deadlines can be missed. However, if no deadlines are missed, then our definition yields priority comparisons that match exactly how EDZL has been specified in prior work. It is possible that other variants could be defined that prioritize jobs differently when deadlines are missed.

Example 3.7. The PD^2 and EPDF Pfair algorithms can also be modeled using our framework. Consider the EPDF algorithm introduced in Example 2.2 in Section 2.1.2. Again, we illustrate assuming jobs are released in a synchronous periodic fashion. First, we represent each task $T_i(e_i,p_i)$ by a task T_i' with $e_i'=1$ and $p_i' = \frac{1}{u_i}$. The EPDF subtask T_i^j then corresponds to the job $T_{i,j}'$. Second, we define the eligibility time of $T_{i,j}'$ as $\epsilon_{i,j} = r_i^j$. Third, we define the prioritization function for job $T_{i,j}'$ as $\chi(T_{i,j},t) = d_i^j$. Note, that $\chi(T_{i,j},t)$ is always an integral number.

This mapping is illustrated in Figure 3.3 using the task set $\tau = \{T_1(3,8), T_2(3,7), T_3(3,6), T_4(1,2)\}$ scheduled on two fully-available processors. Inset (a) shows an EPDF schedule for τ. Subtask windows are shown in bold. Inset (b) shows a schedule for τ', which is constructed from τ in the way described above. In this figure, the release time of each job $T_{i,j}'$ is denoted by an up arrow and its deadline is
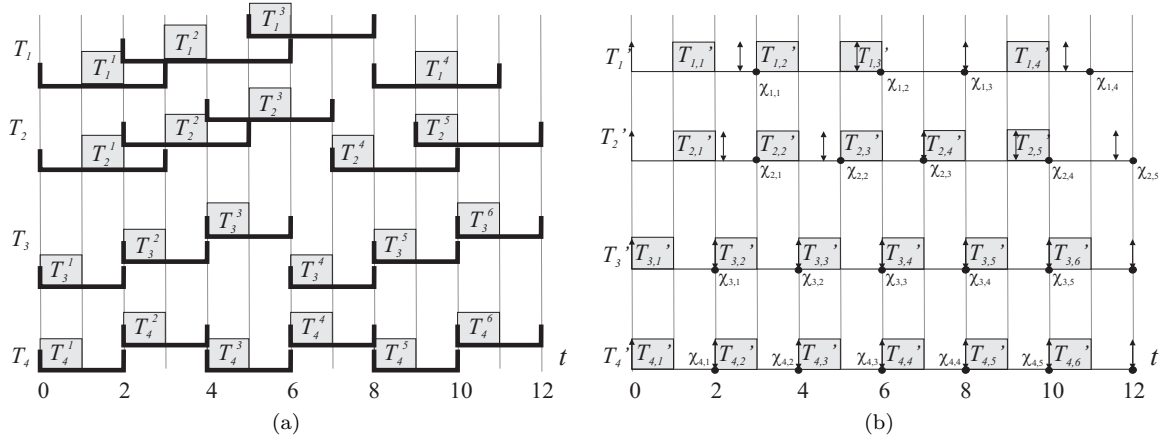
Figure 3.3: **(a)** An EPDF schedule for the task set τ from Example 3.7. **(b)** Equivalent schedule obtained using prioritization functions.

denoted by a down arrow. χ-values are depicted as black circles.

PD^2 differs from EPDF in that two special tie-breaking rules are used in the event of a deadline tie. We can capture the effects of these tie breaks by slightly shifting the value of a job's prioritization function and letting it be non-integral.

3.3   Tardiness Bound

In this section, we show that any scheduling algorithm (specified according to Definition 3.1) has bounded tardiness if its prioritization functions are "window-constrained," as defined below in Definition 3.4. This definition imposes two separate constraints on χ-values. We show that if either is violated, then tardiness may become unbounded. In this section, we consider a system with partially available processors; later, in Section 3.4, we consider the special case when all processors are fully available as well as some other extensions to the analysis.

3.3.1   Definitions

The system start time is assumed to be zero. For any time $t > 0$, t^- denotes the time $t - v$ in the limit $v \to 0+$.

Definition 3.2. (pending jobs) $T_{i,j}$ is *pending* at time t in a schedule $\mathcal{S}$ if $T_{i,j}$ is eligible at time t and $T_{i,j}$ has not completed execution by t in $\mathcal{S}$.

Definition 3.3. (ready jobs) A pending job $T_{i,j}$ is *ready* at time t in a schedule $\mathcal{S}$ if all prior jobs of T_i have completed execution by t in $\mathcal{S}$.

47

Definition 3.4. (window-constrained priorities) A scheduling algorithm's prioritization functions are *window-constrained* iff, for each task T_i, there exist constants ϕ_i and ψ_i such that, for each job $T_{i,j}$ of T_i and time t,

$$r_{i,j} - \phi_i \leq \chi(T_{i,j}, t) \leq d_{i,j} + \psi_i. \tag{3.3}$$

Note that (3.3) requires a job's χ-values to lie within a window $[r_{i,j} - \phi_i, d_{i,j} + \psi_i]$ that is defined with respect to its release time and deadline. Note also that the constants ϕ_i and ψ_i may be positive or negative; however, if negative, the interval $[r_{i,j} - \phi_i, d_{i,j} + \psi_i]$ cannot be empty.

It is easy to see that, other than RM, all of the algorithms considered in Section 3.2 have prioritization functions that satisfy (3.3). In contrast, the prioritization function specified for RM fails to be window-constrained because it violates the required lower bound: as new jobs of each task T_i are released, $\chi(T_{i,j}, t) < r_{i,j} - \phi_i$ will eventually hold for some job $T_{i,j}$ for any choice of the constant ϕ_i. It can be shown that the task system in Example 3.2 has unbounded tardiness. In particular, if the job-release pattern in Figure 3.1(b) recurs repeatedly, then the processing capacity available to T_4 every 12 time units is the same as is depicted in Figure 3.1(b). This capacity is less than the amount of work generated by T_4 during the same interval. As a result, more and more work shifts to future intervals, causing tardiness for T_4 to grow unboundedly. (The fact that tardiness can be unbounded under RM was also established by Devi (2006).)

It is possible to "fix" the prioritization functions for RM so that the required lower bounds are adhered to, but then the upper bounds will be violated. For example, we could simply define $\chi(T_{i,j}, t) = i + t'$, where t' is the time where the most recent job release occurred at or before t. This definition simply shifts the χ-values defined earlier to future points in time as new jobs are released. However, we know that tardiness for T_4 in Example 3.2 is unbounded, so eventually $\chi(T_{4,j}, t) > d_{4,j} + \psi_4$ will hold for some pending job $T_{4,j}$ of T_4 for any choice of the constant ψ_4. Intuitively, Inequality (3.3) ensures that any job $T_{i,j}$ eventually becomes the highest-priority job in the system and will execute until completion. We summarize this discussion as follows. (Recall that any task set considered in this chapter is assumed to satisfy (3.1).)

Observation: *If either the lower or upper bound given in (3.3) is eliminated, then there exists a prioritization scheme that satisfies the remaining condition for which tardiness is unbounded for some task set.*

48
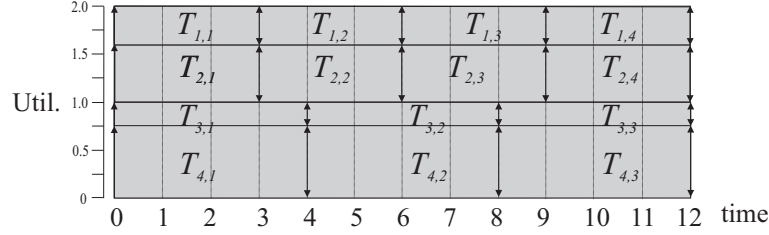
Figure 3.4: PS schedule for τ in Example 3.1.

Most of the rest of this chapter is devoted to showing that any scheduling algorithm $\mathcal{A}$ with window-constrained prioritization functions has bounded tardiness. The tardiness bound established for $\mathcal{A}$ is derived by comparing the allocations to a concrete task system τ in an ideal processor-sharing (Processor-Sharing (PS)) schedule to those in a schedule produced by $\mathcal{A}$. (We remind the reader that, in a concrete task system, job release times, eligibility times, deadlines, and execution times are specified — see Definition 1.3 in Section 1.5.)

In a *PS schedule*, each job of a task T_i is executed at a constant rate of $u_{i,j} = \frac{e_{i,j}}{p_i} \leq u_i$ between its release and deadline (Stoica et al., 1996). Figure 3.4 depicts an example. In this figure, the execution of each job $T_{i,j}$ is represented as a rectangle of length $p_i = d_{i,j} - r_{i,j}$ and height $u_{i,j}$. Therefore, the allocation of each job between its release time and deadline in this schedule is $u_{i,j} \cdot p_i = e_{i,j}$.

Note that a PS schedule does not depend on processor availability. Also, in such a schedule, each job completes exactly at its deadline. Thus, if a job misses its deadline, then it is "lagging behind" the corresponding PS schedule — this concept of "lag" is instrumental in the analysis and is formalized below. (A similar lag-based analysis was used by Devi and Anderson (2008b) to establish tardiness bounds for preemptive and non-preemptive global EDF).

Definition 3.5. Let $\mathsf{A}(T_{i,j}, t_1, t_2, \mathcal{S})$ be the allocation of job $T_{i,j}$ during the interval $[t_1, t_2)$ in an arbitrary schedule $\mathcal{S}$. Let $\mathsf{A}(T_i, t_1, t_2, \mathcal{S})$ be the allocation of task T_i during the interval $[t_1, t_2)$ in the schedule $\mathcal{S}$.

The difference between the allocations to $T_{i,j}$ up to time t in a PS schedule $\mathcal{PS}$ and an arbitrary schedule $\mathcal{S}$, termed the *lag of $T_{i,j}$ at time t in schedule $\mathcal{S}$*, is given by

$$\mathsf{lag}(T_{i,j}, t, \mathcal{S}) = \mathsf{A}(T_{i,j}, 0, t, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t, \mathcal{S}). \tag{3.4}$$

Task lags can be similarly defined:

$$\mathsf{lag}(T_i, t, \mathcal{S}) = \sum_{j \geq 1} \mathsf{lag}(T_{i,j}, t, \mathcal{S}) = \sum_{j \geq 1} \mathsf{A}(T_{i,j}, 0, t, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t, \mathcal{S}). \tag{3.5}$$

49

Finally, the *lag for a finite job set* Φ *at time* t *in the schedule* $\mathcal{S}$ is defined by

$$\mathsf{LAG}(\Phi, t, \mathcal{S}) = \sum_{T_{i,j} \in \Phi} \mathsf{lag}(T_{i,j}, t, \mathcal{S}) = \sum_{T_{i,j} \in \Phi} (\mathsf{A}(T_{i,j}, 0, t, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t, \mathcal{S})). \tag{3.6}$$

Since $\mathsf{LAG}(\Phi, 0, \mathcal{S}) = 0$, the following holds for $t' \leq t$.

$$\mathsf{LAG}(\Phi, t, \mathcal{S}) = \mathsf{LAG}(\Phi, t', \mathcal{S}) + \mathsf{A}(\Phi, t', t, \mathcal{PS}) - \mathsf{A}(\Phi, t', t, \mathcal{S}) \tag{3.7}$$

The concept of lag is important because, if lags remain bounded, then tardiness is bounded as well.

Definition 3.6. A time interval $[t_1, t_2)$ is *busy* for a job set Φ in schedule $\mathcal{S}$ if, at each time $t \in [t_1, t_2)$, all m processors execute jobs from Φ in this schedule, and is *non-busy* for Φ otherwise.

When using the above terminology, we will omit "for Φ" if the job set under consideration is clear. According to the lemma below, the lag for a job set Φ cannot increase across a busy interval for Φ. This fact was proved in the context of global EDF in (Devi et al., 2006). However, since the proof relies only on the fact that the interval in question is busy, and not on how jobs are scheduled, it applies in our context as well. Later, we will examine the behavior of the LAG function over an interval where some processors are unavailable.

Lemma 3.1. *For any interval* $[t_1, t_2)$ *that is busy for* Φ, $\mathsf{LAG}(\Phi, t_2, \mathcal{S}) \leq \mathsf{LAG}(\Phi, t_1, \mathcal{S})$.

Proof. By (3.7),

$$\mathsf{LAG}(\Phi, t_2, \mathcal{S}) = \mathsf{LAG}(\Phi, t_1, \mathcal{S}) + \mathsf{A}(\Phi, t_1, t_2, \mathcal{PS}) - \mathsf{A}(\Phi, t_1, t_2, \mathcal{S}). \tag{3.8}$$

Because the interval $[t_1, t_2)$ is busy, m processors execute jobs from Φ throughout the interval, and thus $\mathsf{A}(\Phi, t_1, t_2, \mathcal{S}) = m \cdot (t_2 - t_1)$. In the ideal PS schedule $\mathcal{PS}$, each job $T_{i,j}$ executes with a constant rate $u_{i,j} \leq u_i$ from its release to its deadline, and thus

$$\mathsf{A}(\Phi, t_1, t_2, \mathcal{PS}) \leq \sum_{T_i \in \tau} \sum_{j > 0} \mathsf{A}(T_{i,j}, t_1, t_2, \mathcal{PS}) \leq \sum_{T_i \in \tau} u_i \cdot (t_2 - t_1) = U_{sum} \cdot (t_2 - t_1).$$

Setting this inequality and $\mathsf{A}(\Phi, t_1, t_2, \mathcal{S}) = m \cdot (t_2 - t_1)$ into (3.8) and applying $U_{sum} \leq \sum_{k=1}^{m} \widehat{u_k} \leq m$, we get
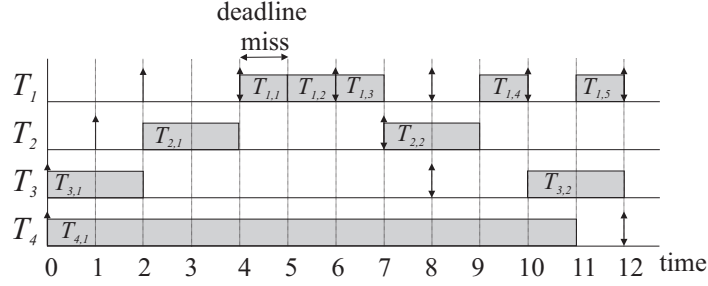
Figure 3.5: A schedule for τ in Example 3.8.

$$\mathsf{LAG}(\Phi, t_2, \mathcal{S}) = \mathsf{LAG}(\Phi, t_1, \mathcal{S}) + \mathsf{A}(\Phi, t_1, t_2, \mathcal{PS}) - \mathsf{A}(\Phi, t_1, t_2, \mathcal{S})$$

$$\leq \mathsf{LAG}(\Phi, t_1, \mathcal{S}) + U_{sum} \cdot (t_2 - t_1) - m \cdot (t_2 - t_1)$$

$$\leq \mathsf{LAG}(\Phi, t_1, \mathcal{S}). \qquad \qquad \square$$

We are interested in non-busy intervals (for a job set) because total lag (for that job set) can increase only across such (non-busy) intervals, and such increases may lead to deadline misses. The following example illustrates how lag can change across busy and non-busy intervals.

Example 3.8. Consider a two-processor system upon which a task set $\tau = \{T_1(1, 2), T_2(2, 6), T_3(2, 8), T_4(11, 12)\}$ is to be scheduled, where the first jobs of T_1, T_2, T_3, and T_4 are released at times 2, 1, 0, and 0 respectively. The total utilization of the system is $U_{sum} = 1/2 + 2/6 + 2/8 + 11/12 = 2$. Assume that both processors are always available, i.e., $\widehat{u_1} = \widehat{u_2} = 1$ and $\sigma_1 = \sigma_2 = 0$, and $\mathcal{A}$ is the FIFO algorithm, i.e., jobs are prioritized using $\chi(T_{i,j}, t) = r_{i,j}$ (assume there are no early releases). Consider the schedule for τ in Figure 3.5. Under $\mathcal{A}$, $T_{1,1}$ misses its deadline at time 4 by one time unit because it cannot preempt $T_{2,1}$ and $T_{4,1}$, which have earlier release times and later deadlines.

Let $\Phi = \{T_{1,1}, \ldots, T_{1,5}, T_{2,1}, T_{3,1}, T_{4,1}\}$ be the set of jobs with deadlines at most 12. The interval $[4, 7)$ in Figure 3.5 is a busy interval for Φ, because all processors execute jobs from Φ throughout the interval. By (3.7), $\mathsf{LAG}(\Phi, 7, \mathcal{S}) = \mathsf{LAG}(\Phi, 4, \mathcal{S}) + \mathsf{A}(\Phi, 4, 7, \mathcal{PS}) - \mathsf{A}(\Phi, 4, 7, \mathcal{S})$, where $\mathcal{S}$ is the schedule under $\mathcal{A}$. The allocation of Φ in the PS schedule $\mathcal{PS}$ during the interval $[4, 7)$ is $\mathsf{A}(\Phi, 4, 7, \mathcal{PS}) = 3 \cdot (u_1 + u_2 + u_3 + u_4) = 3/2 + 6/6 + 6/8 + 33/12 = 6$. The allocation of Φ in $\mathcal{S}$ throughout $[4, 7)$ is also 6. Thus, $\mathsf{LAG}(\Phi, 7, \mathcal{S}) = \mathsf{LAG}(\Phi, 4, \mathcal{S})$.

Now let $\Phi = \{T_{1,1}\}$ be the set of jobs with deadlines at most 4. Because the jobs $T_{2,1}$ and $T_{4,1}$, which have deadlines after time 4, execute within the interval $[2, 4)$ in Figure 3.5, this interval is non-busy for Φ

in $\mathcal{S}$. By (3.6), $\mathsf{LAG}(\Phi, 4, \mathcal{S}) = \mathsf{A}(\Phi, 0, 4, \mathcal{PS}) - \mathsf{A}(\Phi, 0, 4, \mathcal{S})$. The allocation of Φ in the PS schedule $\mathcal{PS}$ throughout the interval $[0, 4)$ is $\mathsf{A}(\Phi, 0, 4, \mathcal{PS}) = 2 \cdot 1/2 = 1$. The allocation of Φ in $\mathcal{S}$ is $\mathsf{A}(\Phi, 0, 4, \mathcal{S}) = 0$. Thus, $\mathsf{LAG}(\Phi, 4, \mathcal{S}) = 1 - 0 = 1$. Figure 3.5 shows that at time 4, $T_{1,1}$ from Φ is pending. This job has unit execution cost, which is equal to the amount of pending work given by $\mathsf{LAG}(\Phi, 4, \mathcal{S})$.

3.3.2 Tardiness Bound for $\mathcal{A}$

In this section, we first state the main result of the chapter as a theorem, and then derive specific tardiness bounds thereby proving the theorem.

Theorem 3.1. *The tardiness of any task T_k under a window-constrained scheduling algorithm $\mathcal{A}$ is bounded, provided $\sum_{T_i \in \tau} u_i \leq m$ and $\sum_{k=1}^{m} \widehat{u_k} - \max(\overline{F} - 1, 0) \cdot \max(u_\ell) - U_L > 0$, where $\overline{F}$ is the number of processors that may not be fully available to τ and U_L is the sum of $\min(|\tau|, m - 1)$ largest total utilizations of tasks in τ.*

Given an arbitrary non-concrete task system τ^N (where the eligibility times and release times of jobs are not specified – see Definition 1.3), we want to determine the maximum tardiness of any job of any task in any concrete instantiation of τ^N scheduled on m processors. The approach for doing this is based on techniques from (Devi and Anderson, 2008b). Let τ be a concrete instantiation of τ^N. First, we order the jobs in the concrete instantiation using the following rule: $T_{i,j} \prec T_{a,b}$ iff $d_{i,j} < d_{a,b}$ or $(d_{i,j} = d_{a,b}) \wedge i < a$.

Let

$$\rho = \max\left(0, \max_{i \neq a}(\psi_a + \phi_i)\right) \quad \text{and} \quad \mu = \max\left(0, \max_{i \neq a}(p_a + \psi_a + \phi_i)\right) \tag{3.9}$$

Let $T_{\ell,q}$ be a job of a task T_ℓ in τ, let $t_d = d_{\ell,q}$, and let $\mathcal{S}$ be a schedule, produced for τ by the scheduling algorithm $\mathcal{A}$. We assume that the schedule $\mathcal{S}$ has the following property.

(P) The tardiness of every job $T_{k,h}$ such that $T_{k,h} \prec T_{\ell,q}$ is at most $x + e_k$, where $x \geq \rho \geq 0$.

Our goal is to determine the smallest $x \geq \rho$ such that the tardiness of $T_{\ell,q}$ remains at most $x + e_\ell$. Such a result would by induction imply a tardiness of at most $x + e_k$ for all jobs of every task $T_k \in \tau$. Because τ is arbitrary, the tardiness bound will hold for every concrete instantiation of τ^N.

The objective is easily met if $T_{\ell,q}$ completes by its deadline, t_d, so assume otherwise. The completion time of $T_{\ell,q}$ then depends on the demand of the jobs that can compete with $T_{\ell,q}$ after t_d and on the amount of available processor time after t_d. Hence, a value for x can be determined via the following steps.

1. Compute an upper bound on the demand for jobs (including $T_{\ell,q}$) that can compete with $T_{\ell,q}$ after t_d.

2. Determine the amount of such demand necessary for the tardiness of $T_{\ell,q}$ to exceed $x + e_\ell$.

3. Determine the smallest $x \geq \rho$ such that the tardiness of $T_{\ell,q}$ is at most $x + e_\ell$ using the upper bound in Step 1 and the necessary condition in Step 2.

To reason about the tardiness of $T_{\ell,q}$, we need to determine how other jobs delay its execution. To do that, we first define a boolean function of two jobs $T_{i,k}$ and $T_{a,b}$ that will allow us to exclude certain jobs from consideration:

$$\mathsf{LP}(T_{i,k}, T_{a,b}) = (\forall\ t\ :\ d_{a,b} + \psi_a < \chi(T_{i,k}, t)). \tag{3.10}$$

Claim 3.1. *If* $\mathsf{LP}(T_{i,k}, T_{a,b})$ *holds for jobs* $T_{i,k}$ *and* $T_{a,b}$, *then* $\chi(T_{a,b}, t) < \chi(T_{i,k}, t)$ *for any time* t.

Proof. We upper bound $\chi(T_{a,b}, t)$ as follows.

$$\chi(T_{a,b}, t)$$

$$\{\text{by (3.3)}\}$$

$$\leq d_{a,b} + \psi_a$$

$$\{\text{by the condition of the claim and (3.10)}\}$$

$$< \chi(T_{i,k}, t) \qquad\qquad \square$$

Claim 3.1 provides a sufficient condition for a job $T_{i,k}$ to have lower priority (a larger χ-value) than that of $T_{a,b}$ at any time and therefore not compete with $T_{a,b}$ for processor time. In the rest of the proof, four job sets, **d**, **DH**, **DLH**, and **DLL**, are considered. **d** and **DH** are defined as follows.

$$\mathbf{d} = \{T_{i,k} :: d_{i,k} \leq d_{\ell,q} = t_d\} \tag{3.11}$$

$$\mathbf{DH} = \{T_{i,k} :: (d_{i,k} > t_d) \wedge (i \neq \ell) \wedge (\exists\ T_{a,b} \in \mathbf{d} : (a \neq i) :: \neg \mathsf{LP}(T_{i,k}, T_{a,b}))\} \tag{3.12}$$

In this notation, **d** and **D** denote, respectively, jobs with deadlines at most and greater than t_d. The letter **H** in **DH** denotes that $T_{i,k}$'s priority at some time *may be higher* than that of a job of different

task in $\mathbf{d}$ (refer to Claim 3.1). Note that, because $d_{\ell,y} \le d_{\ell,q} = t_d$,

$$(\forall y : y \le q :: T_{\ell,y} \in \mathbf{d}). \tag{3.13}$$

The remaining two job sets are defined as follows.

$$\mathbf{DLH} = \{T_{i,k} :: (d_{i,k} > t_d) \wedge (i \ne \ell) \wedge (\forall\ T_{a,b} \in \mathbf{d} : (a \ne i) :: \mathsf{LP}(T_{i,k}, T_{a,b}))$$
$$\wedge\ (\exists\ T_{a,b} \in \mathbf{DH} : (a \ne i) :: \neg\mathsf{LP}(T_{i,k}, T_{a,b}))\} \tag{3.14}$$

$$\mathbf{DLL} = \{T_{i,k} :: (d_{i,k} > t_d) \wedge (i \ne \ell) \wedge (\forall\ T_{a,b} \in \mathbf{d} : (a \ne i) :: \mathsf{LP}(T_{i,k}, T_{a,b}))$$
$$\wedge\ (\forall\ T_{a,b} \in \mathbf{DH} : (a \ne i) :: \mathsf{LP}(T_{i,k}, T_{a,b}))\} \tag{3.15}$$

If $T_{i,k}$ is in $\mathbf{DLH}$ or $\mathbf{DLL}$, then, for each job $T_{a,b} \in \mathbf{d}$ such that $a \ne i$, $\mathsf{LP}(T_{i,k}, T_{a,b})$ holds, and hence, $T_{i,k}$'s priority *is always lower* than that of any job in $\mathbf{d}$ of a different task. The second letter $\mathbf{L}$ in $\mathbf{DLH}$ and $\mathbf{DLL}$ is intended to denote this. Similarly, the third letter $\mathbf{H}$ in $\mathbf{DLH}$ denotes that job $T_{i,k}$'s priority may be higher than that of a job of a different task T_a that belongs to $\mathbf{DH}$. Finally, the third letter $\mathbf{L}$ in $\mathbf{DLL}$ denotes that job $T_{i,k}$'s priority is always lower than that of any job of a different task T_a that belongs to $\mathbf{DH}$.

Example 3.9. Consider the task set $\tau = \{T_1(1, 2), T_2(1.5, 3), T_3(5, 5)\}$ and the PS schedule for it in Figure 3.6. Job $T_{1,1}$ is released at time 1, and jobs $T_{2,1}$ and $T_{3,1}$ are released at time 0. Consider the job $T_{\ell,q} = T_{1,1}$, which has a deadline at time 3. Assume that there are no early releases and jobs are prioritized as follows. For task T_1, $\chi(T_{1,j}, t) = d_{1,j}$ for all j. For task T_2, $\chi(T_{2,j}, t) = r_{2,j}$ if j is even and $\chi(T_{2,j}, t) = d_{2,j}$ if j is odd. For task T_3, $\chi(T_{3,j}, t) = r_{3,j}$ for all j.

We thus have, $\phi_1 = -p_1$, $\phi_2 = \phi_3 = 0$, $\psi_1 = 0$, $\psi_2 = 0$, and $\psi_3 = -p_3$. With respect to $T_{1,1}$, the four sets mentioned above are $\mathbf{d} = \{T_{1,1}, T_{2,1}\}$, $\mathbf{DH} = \{T_{3,1}, T_{2,2}\}$, $\mathbf{DLH} = \{T_{3,2}\}$, and $\mathbf{DLL} = \{T_{2,3}, T_{2,4}, T_{3,3}\}$. The job $T_{2,2} \in \mathbf{DH}$ because $\chi(T_{2,2}, t) = r_{2,2} = 3 \le d_{1,1} = 3$, and hence, $\mathsf{LP}(T_{2,2}, T_{1,1})$ does not hold. The job $T_{3,2} \in \mathbf{DLH}$ because $\chi(T_{3,2}, t) = r_{3,2} = 5 \le d_{2,2} = 6$, and hence, $\mathsf{LP}(T_{3,2}, T_{2,2})$ does not hold. $\mathbf{DLL}$ would also include any jobs of tasks other than T_1 released after time 12.

We now prove some important relationships between the priorities of jobs in the four sets mentioned above.

Lemma 3.2. *If $T_{a,b} \in \mathbf{DH}$ and $T_{i,k} \in \mathbf{DLL}$, where $a \ne i$, then $\chi(T_{a,b}, t) < \chi(T_{i,k}, t)$ for any time t.*
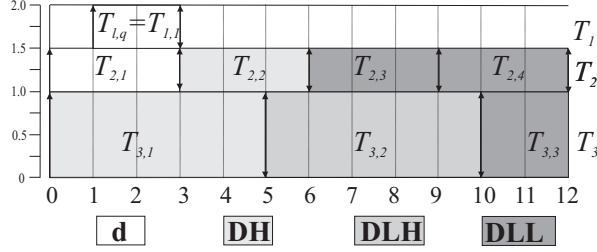
Figure 3.6: Job set partitioning.

Proof. If $T_{i,k}$ in **DLL**, then, by (3.15), $(\forall T_{a,b} \in \mathbf{DH} : (a \neq i) :: \mathsf{LP}(T_{i,k}, T_{a,b}))$. By the condition of the lemma, this implies that $\mathsf{LP}(T_{i,k}, T_{a,b})$ holds. The required result follows from Claim 3.1. □

Lemma 3.3. *If* $T_{a,b} \in \mathbf{d}$ *and* $T_{i,k} \in \mathbf{DLL} \cup \mathbf{DLH}$, *where* $a \neq i$, *then* $\chi(T_{a,b}, t) < \chi(T_{i,k}, t)$ *for any time* t.

Proof. If $T_{i,k} \in \mathbf{DLL} \cup \mathbf{DLH}$, then, by (3.14) and (3.15), $(\forall T_{a,b} \in \mathbf{d} : (a \neq i) :: \mathsf{LP}(T_{i,k}, T_{a,b}))$ holds. By the condition of the lemma, this implies that $\mathsf{LP}(T_{i,k}, T_{a,b})$ holds. The required result follows from Claim 3.1. □

Lemma 3.4. *If a job* $T_{i,k} \in \mathbf{DLL}$ *is scheduled at time* t *or there is an idle available processor at time* t, *and* $T_{a,b} \in \mathbf{d} \cup \mathbf{DH}$ *is ready at time* t, *where* $a \neq i$, *then* $T_{a,b}$ *is scheduled at time* t.

Proof. The case when an available processor is idle at time t is trivial so suppose that this is not the case. If $T_{i,k}$ and $T_{a,b}$ are defined as in the statement of the lemma, and $T_{i,k}$ is scheduled at time t, then $T_{a,b}$ is scheduled at time t as well since, by Lemmas 3.2 and 3.3, $\chi(T_{a,b}, t) < \chi(T_{i,k}, t)$. □

Lemma 3.5. *If a job* $T_{i,k} \in \mathbf{DLH} \cup \mathbf{DLL}$ *is scheduled at time* t *and* $T_{a,b} \in \mathbf{d}$ *is ready at time* t, *where* $a \neq i$, *then* $T_{a,b}$ *is scheduled at time* t.

Proof. If $T_{i,k}$ and $T_{a,b}$ are defined as in the statement of the lemma, and $T_{i,k}$ is scheduled at time t, then $T_{a,b}$ is scheduled as well, since by Lemma 3.3, $\chi(T_{a,b}, t) < \chi(T_{i,k}, t)$. □

Corollary 3.1. *If a job* $T_{i,k} \in \mathbf{DLH} \cup \mathbf{DLL}$ *is scheduled at time* $t \geq t_d$ *and job* $T_{\ell,q}$ *is pending at time* t, *then* T_ℓ *is scheduled at* t.

Proof. If $T_{\ell,q}$ is pending at time $t \geq t_d$, then the earliest pending job of T_ℓ, $T_{\ell,y}$, where $y \leq q$ is ready at time t. The required result follows from (3.13) and Lemma 3.5. □

Determining an upper bound on competing demand. We are now ready to establish the upper bound mentioned in the first step of the proof outline given earlier as a function of job sets **d**, **DH**, **DLH**, and **DLL**.

Definition 3.7. Let $W(\alpha)$ be the total allocation of jobs in the set α in schedule $\mathcal{S}$ after time t_d while job $T_{\ell,q}$ is pending.

We are interested in the allocation of jobs in $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ because these jobs may delay the execution of $T_{\ell,q}$. (By Lemma 3.4, jobs in $\mathbf{DLL}$ cannot delay $T_{\ell,q}$ or prior jobs of T_ℓ.) Their allocation after t_d while $T_{\ell,q}$ is pending, is

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) = W(\mathbf{d}) + W(\mathbf{DH} \cup \mathbf{DLH}). \tag{3.16}$$

Because jobs from $\mathbf{d}$ have deadlines at most t_d, they do not execute in the PS schedule $\mathcal{PS}$ beyond t_d. Thus, the allocation of jobs in $\mathbf{d}$ after time t_d is upper-bounded by the amount of pending work due to jobs in this set at time t_d as given by $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$, which must be positive in order for $T_{\ell,q}$ to miss its deadline at t_d (by (3.13)). Therefore,

$$W(\mathbf{d}) \leq \mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}). \tag{3.17}$$

From (3.16) and (3.17), we have

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) \leq \mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) + W(\mathbf{DH} \cup \mathbf{DLH}). \tag{3.18}$$

Thus, an upper bound on $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH})$ can be obtained by determining bounds for $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$ and $W(\mathbf{DH} \cup \mathbf{DLH})$ individually.

Upper bound on $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$. In deriving this bound, we assume that all busy and non-busy intervals considered are with respect to $\mathbf{d}$ and the schedule $\mathcal{S}$ is produced by the scheduling algorithm $\mathcal{A}$.

To begin, note that, by Lemma 3.1, if no non-busy interval exists in $[0, t_d)$, then $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) \leq \mathsf{LAG}(\mathbf{d}, 0, \mathcal{S}) = 0$. In that which follows, we consider the more interesting case wherein some non-busy interval exists in $[0, t_d)$. An interval could be non-busy for two reasons:

1. There are not enough ready jobs in $\mathbf{d}$ to occupy all available processors, so it is immaterial whether jobs from $\mathbf{DH}$, $\mathbf{DLH}$, or $\mathbf{DLL}$ execute during the interval.

2. There are tasks with ready jobs in $\mathbf{d}$ that cannot execute because, within certain sub-intervals, some processors are not available (because of capacity restrictions) or jobs in $\mathbf{DH}$ occupy one or

more processors because they have higher priority. Note that, by Lemma 3.5, jobs in **DLH** and **DLL** cannot execute at time instants when there are ready unscheduled jobs in **d**.

Jobs with deadlines after time t_d may prevent the execution of jobs in **d** before time t_d (if such jobs become eligible before t_d) and hence increase the LAG for **d**.

Definition 3.8. ($\tau_{\mathbf{DH}}$) Let $\tau_{\mathbf{DH}}$ be the set of tasks that have jobs in **DH**.

Definition 3.9. (δ_i) Let δ_i be the total allocation of task T_i's jobs in **DH** in the schedule $\mathcal{S}$ by time t_d.

In much of the rest of the analysis, we focus on a time t_n defined as follows.

Definition 3.10. If there exists a time instant t such that there are at most $m-1$ tasks with ready jobs in **d** at time t^- and all these tasks execute at time t^-, then define t_n to be the latest such time instant at or before t_d; if no such t exists, then let $t_n = 0$.

We express a bound on $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$ in terms of individual task parameters and processor availability functions using Lemmas 3.6, 3.7, and 3.8, which are proved in an appendix. Lemma 3.7 establishes a relationship between $\mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S})$ and $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$. Lemmas 3.6 and 3.8 were initially proved in (Devi et al., 2006) in the context of global EDF, for the case where all processors are fully available. The proof of each lemma relies only on Property (P) and, for Lemma 3.7, the definition of t_n. In particular, the exact way in which jobs are scheduled does not arise.

Lemma 3.6: $\mathsf{lag}(T_k, t, \mathcal{S}) \leq x \cdot u_k + e_k$ for any task T_k and $t \in [0, t_d]$.

Lemma 3.7: $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) \leq \mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k$.

Definition 3.11. ($U(\tau, y)$ and $E(\tau, y)$) Let $U(\tau, y)$ $(E(\tau, y))$ be the set of at most $\min(|\tau|, y)$ tasks from τ of highest utilization (execution cost), where $|\tau|$ is the number of tasks in τ, and let

$$E_L = \sum_{T_i \in E(\tau, m-1)} e_i \quad \text{and}$$

$$U_L = \sum_{T_i \in U(\tau, m-1)} u_i.$$

Lemma 3.8: $\mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) \leq E_L + x \cdot U_L$.

Using Lemmas 3.7 and 3.8, we can upper bound $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S})$ in (3.18).

Lemma 3.9: $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) \leq E_L + x \cdot U_L + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k$.

Upper bound on $W(\mathbf{DH} \cup \mathbf{DLH})$. The jobs in $\mathbf{DH} \cup \mathbf{DLH}$ may delay the execution of $T_{\ell,q}$ because some of these jobs may have higher priority than $T_{\ell,q}$ at some time. We now upper-bound the total execution demand due to jobs in $\mathbf{DH} \cup \mathbf{DLH}$. Lemmas 3.10 and 3.11, which are proved in the appendix, upper-bound the release times of jobs in $\mathbf{DH} \cup \mathbf{DLH}$ using ρ and μ from (3.9).

Lemma 3.10. *If $T_{i,k} \in \mathbf{d} \cup \mathbf{DH}$, then $r_{i,k} \leq t_d + \rho$.*

Lemma 3.11. *If $T_{i,k} \in \mathbf{DLH}$, then $r_{i,k} \leq t_d + \rho + \mu$.*

Similarly to Definition 3.8, we define the following task set.

Definition 3.12. ($\tau_{\mathbf{DLH}}$) Let $\tau_{\mathbf{DLH}}$ be the set of tasks that have jobs in $\mathbf{DLH}$.

Lemma 3.12. *Task $T_i \in \tau_{\mathbf{DH}}$ can have at most $\left\lceil \frac{\rho}{p_i} \right\rceil$ jobs in $\mathbf{DH}$ with release times after t_d. Task $T_i \in \tau_{\mathbf{DLH}}$ can have at most $\left\lceil \frac{\rho+\mu}{p_i} \right\rceil$ jobs in $\mathbf{DLH}$ with release times after t_d.*

Proof. Suppose that $T_{i,k} \in \mathbf{DH} \cup \mathbf{DLH}$ and $r_{i,k} > t_d$. If $T_{i,k} \in \mathbf{DH}$, then, by Lemma 3.10, $r_{i,k} \leq t_d + \rho$. If $T_{i,k} \in \mathbf{DLH}$, then, by Lemma 3.11, $r_{i,k} \leq t_d + \rho + \mu$. Because task T_i's consecutive job releases are separated by at least p_i time units, the lemma follows. $\qquad\square$

Lemma 3.13: $W(\mathbf{DH} \cup \mathbf{DLH}) \leq \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left(\left\lceil \frac{\rho+\mu}{p_i} \right\rceil + 1 \right) \cdot e_i \right) - \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i$

Proof. Consider $T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}$. Each job $T_{i,k}$ in $\mathbf{DH} \cup \mathbf{DLH}$ is released either at or before t_d or after t_d. Because each job in $\mathbf{DH} \cup \mathbf{DLH}$ has a deadline after t_d, each T_i has at most one job in $\mathbf{DH} \cup \mathbf{DLH}$ with a release time at or before t_d. The demand due to this job is at most e_i. By Lemma 3.12, the demand of jobs of T_i in $\mathbf{DH} \cup \mathbf{DLH}$ released after t_d is at most $\left\lceil \frac{\rho+\mu}{p_i} \right\rceil \cdot e_i$. The allocation of task T_i's jobs in $\mathbf{DH}$ in schedule $\mathcal{S}$ before time t_d is δ_i, by Definition 3.9. Thus, the allocation of all jobs in $\mathbf{DH} \cup \mathbf{DLH}$ after time t_d in schedule $\mathcal{S}$ while $T_{\ell,q}$ is pending is

$$
\begin{aligned}
W(&\mathbf{DH} \cup \mathbf{DLH}) \\
&\leq \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left\lceil \frac{\rho+\mu}{p_i} \right\rceil \cdot e_i + e_i \right) - \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i \\
&= \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left(\left\lceil \frac{\rho+\mu}{p_i} \right\rceil + 1 \right) \cdot e_i \right) - \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i \qquad \square
\end{aligned}
$$

Upper bound on $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH})$.

Definition 3.13. Let $\alpha(\tau, \ell) \geq \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left(\left\lceil \frac{\rho+\mu}{p_i} \right\rceil + 1 \right) \cdot e_i \right)$ be a scheduling-algorithm-dependent bound on the competing demand due to jobs in $\mathbf{DH}$ and $\mathbf{DLH}$.

From (3.18), Lemma 3.9, and Lemma 3.13 we have

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH})$$

$$\{\text{by (3.18)}\}$$

$$\leq \mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) + W(\mathbf{DH} \cup \mathbf{DLH})$$

$$\{\text{by Lemmas 3.9 and 3.13}\}$$

$$\leq E_L + x \cdot U_L + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k$$

$$+ \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left(\left\lceil \frac{\rho + \mu}{p_i} \right\rceil + 1 \right) \cdot e_i \right) - \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i$$

$$= E_L + x \cdot U_L + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \sum_{T_i \in \tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}}} \left(\left(\left\lceil \frac{\rho + \mu}{p_i} \right\rceil + 1 \right) \cdot e_i \right)$$

$$\{\text{by Definition 3.13}\}$$

$$\leq E_L + x \cdot U_L + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \alpha(\tau, \ell) \tag{3.19}$$

Claim 3.2. *The expression* $\sum_{T_i \in \tau \setminus T_\ell} \left(\left(\left\lceil \frac{\rho + \mu}{p_i} \right\rceil + 1 \right) \cdot e_i \right)$ *(conservatively) upper-bounds* $\alpha(\tau, \ell)$ *for any window-constrained scheduler.*

Proof. The claim follows from $\tau_{\mathbf{DH}} \cup \tau_{\mathbf{DLH}} \subseteq \tau \setminus T_\ell$. $\qquad\square$

In Section 3.4, we will discuss how to compute tighter bounds for $\alpha(\tau, \ell)$ for GEDF and FIFO schedulers.

Necessary condition for tardiness to exceed $x + e_\ell$. We now find the amount of competing work that is necessary for $T_{\ell,q}$ to miss its deadline by more than $x + e_\ell$ time units. Job $T_{\ell,q}$'s tardiness depends on the amount of competing demand $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH})$ and on the amount of processor time available to τ after time t_d.

Definition 3.14. Let $\beta_k^* \geq \beta_k^l(x + e_\ell)$ be the amount of processor time available to tasks in τ during the interval $[t_d, t_d + x + e_\ell)$ on processor k in schedule $\mathcal{S}$. Let $R = \sum_{k=1}^{m}(x + e_\ell - \beta_k^*)$ be the total amount of processor time that is *not available* to τ during $[t_d, t_d + x + e_\ell)$.

In the rest of this dissertation the following definition will be used.

Definition 3.15. Let F be the number of processors that are fully available, i.e., $F = |k :: \beta_k^l(\Delta) = \Delta|$. Let $\overline{F} = m - F$ be the number of processors that may not be fully available.

Lemma 3.14. *If at most F tasks with ready jobs in $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ are scheduled at time $t^* \in [t_d + \rho, t_d + x + e_\ell)$, $T_{\ell,q}$ is pending at t^*, and there is an idle available processor at time t^* or a job from $\mathbf{DLL}$ is scheduled at time t^*, then* **(i)** *task T_ℓ is scheduled at t^*, and* **(ii)** *T_ℓ is guaranteed uninterrupted execution until the job $T_{\ell,q}$ completes.*

Proof. **(i)** follows from Corollary 3.1. To prove **(ii)**, assume that the antecedent of the lemma holds. Let $A(t)$ $(B(t))$ be the number of tasks that have ready jobs in $\mathbf{d}$ $(\mathbf{DH})$ at time $t \geq t^*$. By Lemma 3.4, all tasks with ready jobs in $\mathbf{d} \cup \mathbf{DH}$ are scheduled at time t^*, and hence,

$$A(t^*) + B(t^*) \leq F. \tag{3.20}$$

Suppose, contrary to the statement of the lemma, that T_ℓ executes uninterruptedly within $[t^*, t')$ but is preempted at time t' so that $T_{\ell,q}$ is pending at t'. By Lemma 3.5, no job in $\mathbf{DLH} \cup \mathbf{DLL}$ can be scheduled at time t' (since $T_{\ell,q} \in \mathbf{d}$). Therefore, at time t', all available processors are occupied by tasks with ready jobs in $\mathbf{d} \cup \mathbf{DH}$, and T_ℓ has ready job (in $\mathbf{d}$) at time t' that is not scheduled. This implies $A(t') + B(t') > F$, and, by (3.20),

$$A(t') + B(t') > A(t^*) + B(t^*). \tag{3.21}$$

By Lemma 3.10, all jobs in $\mathbf{d} \cup \mathbf{DH}$ are released at or before $t_d + \rho$. Therefore, the number of tasks with ready jobs in $\mathbf{d} \cup \mathbf{DH}$ at time $t' > t^*$, $A(t') + B(t')$, cannot be higher than $A(t^*) + B(t^*)$, i.e., $A(t') + B(t') \leq A(t^*) + B(t^*)$. This contradicts (3.21). $\square$

The following lemma establishes a lower bound on the competing demand for $T_{\ell,q}$.

Lemma 3.15. *If the tardiness of $T_{\ell,q}$ exceeds $x + e_\ell$, where $x \geq \rho$, then*

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R > (m - (m - a) \cdot u_\ell) \cdot x + (1 - a) \cdot \rho + e_\ell, \tag{3.22}$$

where $a = \min(m, F + 1)$.

Proof. Assume that

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R \leq (m - (m - a) \cdot u_\ell) \cdot x + (1 - a) \cdot \rho + e_\ell \tag{3.23}$$

holds and suppose, contrary to the statement of the lemma, that

(T) the tardiness of $T_{\ell,q}$ exceeds $x + e_\ell$.

In the rest of the proof, we say that a time instant $t \geq t_d$ (or an interval) is *WR-occupied* if each processor either executes a job from $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ or is unavailable; otherwise, we say that t is *WR-free*. The prefix *"WR"* denotes that all processors contribute to the allocation $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R$. If the time instant $t \geq t_d$ is *WR*-free, then either at least one available processor is idle at t, or a job from $\mathbf{DLL}$ is scheduled at time t. Because, by (T), $T_{\ell,q} \in \mathbf{d}$ is pending throughout the interval $[t_d, t_d + x + e_\ell)$, the following property holds by Corollary 3.1:

(E) task T_ℓ executes at each *WR*-free instant within $[t_d, t_d + x + e_\ell)$.

By (P), the preceding job $T_{\ell,q-1}$ (if it exists) completes by time

$$t' \leq t_d - p_\ell + e_\ell + x \leq t_d + x. \tag{3.24}$$

Thus, $t_d + x$ is the latest time at which $T_{\ell,q}$ may become ready. If the latest *WR*-occupied instant in the interval $[t_d, t_d + x + e_\ell)$ is at or before $t_d + x$, then, by (E), $T_{\ell,q}$ executes uninterruptedly after $t_d + x$ and its tardiness is at most $x + e_{\ell,q} \leq x + e_\ell$, contrary to (T). In the rest of the proof, we assume that the latest *WR*-occupied instant in the interval $[t_d, t_d + x + e_\ell)$ is after $t_d + x$.

Suppose that at most F processors execute jobs from $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ at some *WR*-free instant $t^* \in [t_d + \rho, t_d + x)$. In this case, because t^* is *WR*-free, some processor is idle or a job in $\mathbf{DLL}$ is scheduled there. Thus, by Lemma 3.14, T_ℓ is guaranteed uninterrupted execution at or after time t^* until $T_{\ell,q}$ finishes. By (3.24), $T_{\ell,q-1}$ (if it exists) finishes its execution by time $t' \leq t_d + x$, so $T_{\ell,q}$ finishes by time $t' + e_{\ell,q} \leq t_d + x + e_{\ell,q} \leq t_d + x + e_\ell$, thereby having tardiness at most $x + e_\ell$, contrary to (T).

In the rest of the proof, we assume the following:

(N) at least $a = \min(m, F + 1)$ processors execute jobs from $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ at each *WR*-free instant in $[t_d + \rho, t_d + x)$.

Let B_1, B_2, and B_3 be the total length of *WR*-occupied intervals within $[t_d, t_d + \rho)$, $[t_d + \rho, t_d + x)$, and $[t_d + x, t_d + x + e_\ell)$, respectively. (Recall, from (P), that $x \geq \rho$.) Let $B = B_1 + B_2 + B_3$. This is illustrated in Fig. 3.7.

We now find a lower bound on B. Suppose first that $B \leq x - x \cdot u_\ell$. In this case, the total length of *WR*-free intervals during $[t_d, t_d + x + e_\ell)$ is $x + e_\ell - B \geq x + e_\ell - (x - x \cdot u_\ell) \geq x \cdot u_\ell + e_\ell$. Thus, by (E), T_ℓ executes for at least $x \cdot u_\ell + e_\ell$ time units after time t_d within the interval $[t_d, t_d + x + e_\ell)$. By Lemma 3.6, the total amount of pending work for T_ℓ at time t_d, including work due to $T_{\ell,q}$, is at
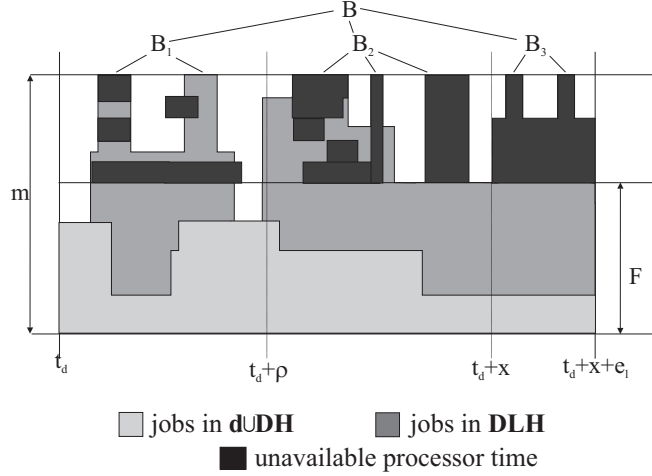
Figure 3.7: Structure of *WR*-occupied intervals in Lemma 3.15.

most $x \cdot u_\ell + e_\ell$, and thus $T_{\ell,q}$ completes by time $t_d + x + e_\ell$ and its tardiness is at most $x + e_\ell$. This contradicts (T). In the rest of the proof, we consider the other possibility, i.e.,

$$B = x - x \cdot u_\ell + v, \tag{3.25}$$

where $v > 0$.

By (E), at least one processor executes a job from **d** at each *WR*-free instant within $[t_d, t_d + \rho)$ (because T_ℓ executes at each such instant). The total length of all *WR*-free intervals within $[t_d, t_d + \rho)$ is

$$L_1 = \rho - B_1. \tag{3.26}$$

By (N), at least a processors execute jobs from $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ at each *WR*-free instant in $[t_d + \rho, t_d + x)$. The total length of all *WR*-free intervals within $[t_d + \rho, t_d + x)$ is $x - \rho - B_2 = x - \rho - (B - B_1 - B_3)$. Thus, the total processor allocation to jobs in $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ in *WR*-free intervals within $[t_d + \rho, t_d + x)$ is at least

$$L_2 = a \cdot (x - \rho - B + B_1 + B_3). \tag{3.27}$$

By (E), at least one processor executes a job from **d** at each *WR*-free instant within $[t_d + x, t_d + x + e_\ell)$ (again, because T_ℓ executes at each such instant). The total length of all *WR*-free intervals in $[t_d + x, t_d + x + e_\ell)$ is

$$L_3 = e_\ell - B_3. \tag{3.28}$$

By (3.25), the sum of the total allocation to jobs in $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ and the unavailable processor

time in all *WR*-occupied intervals in $[t_d, t_d + x + e_\ell)$ is

$$L_b = m \cdot B = m \cdot (x - x \cdot u_\ell + v). \tag{3.29}$$

Let Z be the total allocation to jobs in $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ within $[t_d, t_d + x + e_\ell)$. Because each processor is either unavailable or executes a job from $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ at every *WR*-occupied instant and at least one processor executes T_ℓ at every *WR*-free instant, summing the lengths of all *WR*-free intervals in $[t_d, t_d + \rho)$ and $[t_d + x, t_d + x + e_\ell)$, given by (3.26) and (3.28), the allocation of jobs in $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$ in *WR*-free intervals within $[t_d + \rho, t_d + x)$, given by (3.27), and the total processor allocation and the unavailable processor time in *WR*-occupied intervals in $[t_d, t_d + x + e_\ell)$, given by (3.29), we have

$$Z + R \geq L_1 + L_2 + L_3 + L_b,$$

where R is defined earlier in Definition 3.14. From the inequality above, we have

$$Z + R \geq L_1 + L_2 + L_3 + L_b$$

$$\{\text{by (3.26), (3.27), (3.28), and (3.29)}\}$$

$$= \rho - B_1 + a \cdot (x - \rho - B + B_1 + B_3) + e_\ell - B_3 + m \cdot (x - x \cdot u_\ell + v)$$

$$\{\text{setting } B' = B_1 + B_3 \text{ and } B = x - x \cdot u_\ell + v, \text{ which follows from (3.25)}\}$$

$$= e_\ell + \rho - B' + a \cdot (x \cdot u_\ell - v - \rho + B') + m \cdot (x - x \cdot u_\ell + v)$$

$$= e_\ell + \rho - B' + a \cdot x \cdot u_\ell - a \cdot v - a \cdot \rho + a \cdot B' + m \cdot x - m \cdot x \cdot u_\ell + m \cdot v$$

$$= e_\ell + (m - (m - a) \cdot u_\ell) \cdot x + (m - a) \cdot v + (a - 1) \cdot B' + (1 - a) \cdot \rho. \tag{3.30}$$

By our assumption at the beginning of the proof, $T_{\ell,q}$'s tardiness exceeds $x + e_\ell$. Because $T_{\ell,q} \in \mathbf{d}$, at time $t_d + x + e_\ell$, there is therefore unfinished work on jobs in $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$. Let $Z' > 0$ be this remaining work. To find Z', we subtract $Z + R$ from $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R$.

$$Z' = W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R - Z - R$$

$$\{\text{by (3.23)}\}$$

$$\leq (m - (m - a) \cdot u_\ell) \cdot x + (1 - a) \cdot \rho + e_\ell - Z - R$$

$$\{\text{by (3.30)}\}$$

63

$$\leq (m - (m - a) \cdot u_\ell) \cdot x + (1 - a) \cdot \rho + e_\ell - e_\ell$$

$$- (m - (m - a) \cdot u_\ell) \cdot x - (m - a) \cdot v - (a - 1) \cdot B' - (1 - a) \cdot \rho$$

$$= (1 - a) \cdot B' - (m - a) \cdot v.$$

By (N), $1 - a = 1 - \min(m, F + 1) = \max(-F, 1 - m) = -\min(F, m - 1) \leq 0$ and $m - a = m - \min(m, F + 1) = \max(m - F - 1, 0) = \max(\overline{F} - 1, 0) \geq 0$, and thus $Z' \leq 0$. Therefore, there is no work pending at time $t_d + x + e_\ell$ for jobs in $\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}$, which implies that $T_{\ell,q}$'s tardiness is at most $x + e_\ell$, contrary to (T). $\qquad\square$

Deriving a tardiness bound. In that which follows, it is more convenient to use the following form of (3.22):

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R > (m - \max(\overline{F} - 1, 0) \cdot u_\ell) \cdot x + \max(\overline{F} - m, 1 - m) \cdot \rho + e_\ell. \qquad (3.31)$$

This expression is obtained from (3.22) by replacing $1 - a$ by $\max(\overline{F} - m, 1 - m)$ and $m - a$ by $\max(\overline{F} - 1, 0)$.

Earlier, in (3.18), we established an upper bound on $W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH})$. Using Definition 3.14, we can upper-bound R as follows.

$$R$$

$$\{\text{by Definition 3.14}\}$$

$$= \sum_{k=1}^{m}(x + e_\ell - \beta_k^*)$$

$$\{\text{by Definition 3.14}\}$$

$$\leq \sum_{k=1}^{m}(x + e_\ell - \beta_k^l(x + e_\ell))$$

$$\{\text{ by } (1.1) \}$$

$$\leq \sum_{k=1}^{m}(x + e_\ell - \widehat{u_k} \cdot (x + e_\ell - \sigma_k)) \qquad (3.32)$$

To this point, x has only been constrained to be at least ρ. We now show that if x is further constrained according to the definition below, then the tardiness of $T_{\ell,q}$ is at most $x + e_\ell$.

Definition 3.16. Let $x = \max(\rho, z)$, where

$$z = \frac{E_L + \max(V(\ell))}{\sum_{k=1}^{m} \widehat{u_k} - \max(\overline{F} - 1, 0) \cdot \max(u_\ell) - U_L}, \tag{3.33}$$

and

$$V(\ell) = e_\ell \cdot \left(\sum_{k=1}^{m} (1 - \widehat{u_k}) - 1 \right) + 2 \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \alpha(\tau, \ell) + \min(m - \overline{F}, m - 1) \cdot \rho. \tag{3.34}$$

Lemma 3.16. *With x as defined in Definition 3.16, the tardiness of $T_{\ell,q}$ is at most $x + e_\ell$ provided the denominator of (3.33) is positive.*

Proof. Suppose that the denominator of (3.33) is positive and, contrary to the statement of the lemma, that the tardiness of $T_{\ell,q}$ exceeds $x + e_\ell$. By (3.19) and (3.32),

$$W(\mathbf{d} \cup \mathbf{DH} \cup \mathbf{DLH}) + R$$

$$\leq E_L + x \cdot U_L + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \alpha(\tau, \ell) + \sum_{k=1}^{m} (x + e_\ell - \widehat{u_k} \cdot (x + e_\ell - \sigma_k))$$

$$= E_L + x \cdot U_L + 2 \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \alpha(\tau, \ell) + x \cdot \left(m - \sum_{k=1}^{m} \widehat{u_k} \right) + e_\ell \cdot \sum_{k=1}^{m} (1 - \widehat{u_k}). \tag{3.35}$$

Since, by our assumption, $T_{\ell,q}$'s tardiness is greater than $x + e_\ell$ and $x \geq \rho$, by Lemma 3.15, (3.31) holds. From (3.35) and (3.31), we have

$$(m - \max(\overline{F} - 1, 0) \cdot u_\ell) \cdot x + \max(\overline{F} - m, 1 - m) \cdot \rho + e_\ell$$

$$< E_L + x \cdot U_L + 2 \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \alpha(\tau, \ell)$$

$$+ x \cdot \left(m - \sum_{k=1}^{m} \widehat{u_k} \right) + e_\ell \cdot \sum_{k=1}^{m} (1 - \widehat{u_k}).$$

Rearranging, we have

$$(m - \max(\overline{F} - 1, 0) \cdot u_\ell) \cdot x - m \cdot x + x \cdot \sum_{k=1}^{m} \widehat{u_k} - x \cdot U_L$$

$$< E_L + 2 \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \alpha(\tau, \ell) - \max(\overline{F} - m, 1 - m) \cdot \rho + e_\ell \cdot \left(\sum_{k=1}^{m} (1 - \widehat{u_k}) - 1 \right),$$

which implies

$$x \cdot \left(\sum_{k=1}^{m} \widehat{u_k} - \max(\overline{F} - 1, 0) \cdot u_\ell - U_L \right)$$

$$< E_L + 2 \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \alpha(\tau, \ell) + \min(m - \overline{F}, m - 1) \cdot \rho + e_\ell \cdot \left(\sum_{k=1}^{m} (1 - \widehat{u_k}) - 1 \right).$$

From this, we have

$$x < \frac{E_L + V(\ell)}{\sum_{k=1}^{m} \widehat{u_k} - \max(\overline{F} - 1, 0) \cdot u_\ell - U_L}$$

$$\leq \max \left(\rho, \frac{E_L + \max(V(\ell))}{\sum_{k=1}^{m} \widehat{u_k} - \max(\overline{F} - 1, 0) \cdot \max(u_\ell) - U_L} \right),$$

where $V(\ell)$ is defined as in Definition 3.16. However, this contradicts the definition of x in Definition 3.16.

$\square$

From the above reasoning, Theorem 3.2 below follows.

Theorem 3.2. *The tardiness of any task T_k under a window-constrained scheduling algorithm $\mathcal{A}$ is at most $x + e_k$, where x is as in Definition 3.16, provided the denominator of (3.33) is positive.*

Theorem 3.1, stated earlier, is a corollary of Theorem 3.2.

3.4 Discussion

In this section, we discuss some implications of Theorem 3.2 and consider some extensions and improvements to the analysis given above, such as tightening the tardiness bound for specific scheduling algorithms and processor configurations.

3.4.1 Relative Deadlines Different from Periods

First, note that, the definition of a prioritization function we have assumed is flexible enough to allow task systems with relative deadlines different from periods to be analyzed. By Theorem 3.2 and the definition of tardiness, each job $T_{i,j}$ is guaranteed to complete within $p_i + e_i + x$ time units after its release time $r_{i,j}$. We thus can compute a maximum tardiness bound with respect to an arbitrary relative deadline.

3.4.2 Implications of Theorem 3.2

The requirement to have the denominator of (3.33) to be positive implicitly restricts the maximum per-task utilization the system is able to accommodate without having unbounded deadline tardiness. (Recall that (3.1) is assumed to hold, and by our task model, $|\tau| = n$.)

Corollary 3.2. *Bounded tardiness is guaranteed if*

    (A) $n \leq F$, *or*

    (B) $\sum_{k=1}^{m} \widehat{u_k} - \max(\overline{F} - 1, 0) \cdot \max(u_\ell) - U_L > 0$, *or*

    (C) $\max(u_\ell) < \frac{\sum_{k=1}^{m} \widehat{u_k}}{\max(\overline{F}-1,0)+\min(m-1,n)}$, *or*

    (D) $m \geq 2$ *and* $F \geq m - 1$.

Proof. (A) follows trivially from the fact that if tasks do not compete for available processors, then no deadlines are missed. (B) ensures that the denominator of (3.33) is positive, and by Theorem 3.2 the tardiness of any task in τ is bounded. To prove (C), suppose that

$$\max(u_\ell) < \frac{\sum_{k=1}^{m} \widehat{u_k}}{\max(\overline{F} - 1, 0) + \min(m - 1, n)}.$$

From this, we get $\sum_{k=1}^{m} \widehat{u_k} > \max(u_\ell) \cdot \max(\overline{F} - 1, 0) + \max(u_\ell) \cdot \min(m - 1, n) \geq \max(u_\ell) \cdot \max(\overline{F} - 1, 0) + U_L$, where the last inequality follows from Definition 3.11. By (B), the required result follows. As for (D), if it holds, then $\max(\overline{F} - 1, 0) = \max(m - F - 1, 0) = 0$. By Definition 3.11 and (3.1), $U_L < \sum_{k=1}^{m} \widehat{u_k}$. The required result follows from (B). $\qquad\qquad\square$

    The conditions of Corollary 3.2 are not necessary. Depending on the processor availability pattern, it may be possible to schedule a task system for which some of the conditions from Corollary 3.2 do not hold yet tardiness is still bounded as the following example illustrates.

Example 3.10. Consider a four-processor system, where the first processor is fully available, and all other processors are available for one time unit every three time units as shown in Figure 3.8(a). For these processors, $\widehat{u_1} = 1$, $\widehat{u_2} = \widehat{u_3} = \widehat{u_4} = 1/3$, $\sigma_1 = 0$, and $\sigma_2 = \sigma_3 = \sigma_4 = 2$. The total processing capacity of the system is $\sum_{k=1}^{4} \widehat{u_k} = 1 + 3 \cdot 1/3 = 2$. Suppose that the task set $\tau = \{T_1(3,3), T_2(3,3)\}$ is scheduled. Applying Corollary 3.2 to this task system, we find that bounded deadline tardiness can be
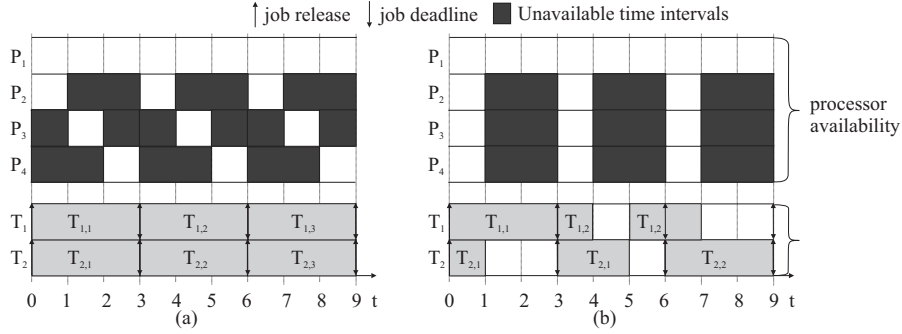
Figure 3.8: Task execution for different processor availability patterns.

guaranteed if

$$\max(u_\ell) < \frac{\sum_{k=1}^m \widehat{u_k}}{\max(\overline{F} - 1, 0) + \min(m - 1, n)}$$

$$= \frac{2}{\max(3 - 1, 0) + \min(3, 2)} = 2/4 = 1/2.$$

Though $\max(u_\ell) = 1 > 1/2$, jobs of T_1 and T_2 always meet their deadlines because at every time instant two processors are available. However, if we attempt to schedule τ on a system with the availability pattern shown in Figure 3.8(b), which is described by the same service functions as the pattern in Figure 3.8(a), we indeed will have unbounded deadline tardiness, because the arriving jobs demand six time units every three time units (assuming the job-arrival pattern continues as shown) but can utilize only four time units.

Uniform multiprocessors. Service functions as defined by (1.1) can also be used to describe a uniform multiprocessor platform, i.e., a platform where processors have different (constant) speeds. Particularly, a service function for which $\sigma_k = 0$ describes a processor with speed $\widehat{u_k} \leq 1$. This can be thought of as a unit-speed processor that is unavailable in infinitesimally small time intervals. The following example illustrates this approximation.

Example 3.11. Consider a processor that is available for two time units every six time units. The amount of available service $\beta^{*[1]}(\Delta)$ is shown in Figure 3.9(a) with a solid line. The service function for this processor is $\beta^{[1]}(\Delta) = \max(0, \widehat{u} \cdot (\Delta - \sigma))$, where $\widehat{u} = 1/3$ and $\sigma = 4$ as shown in Figure 3.9(a). The superscript "[1]" denotes that this is a first approximation of a processor with speed 1/3. It is possible to make processor availability more even, so that the processor is available for one time units every three time units. The respective service curves, $\beta^{*[2]}(\Delta)$ and $\beta^{[2]}(\Delta) = \max(0, 1/3 \cdot (\Delta - 2))$, are shown in Figure 3.9(b). Continuing this process, we can approximate a processor with speed 1/3 by using the

68
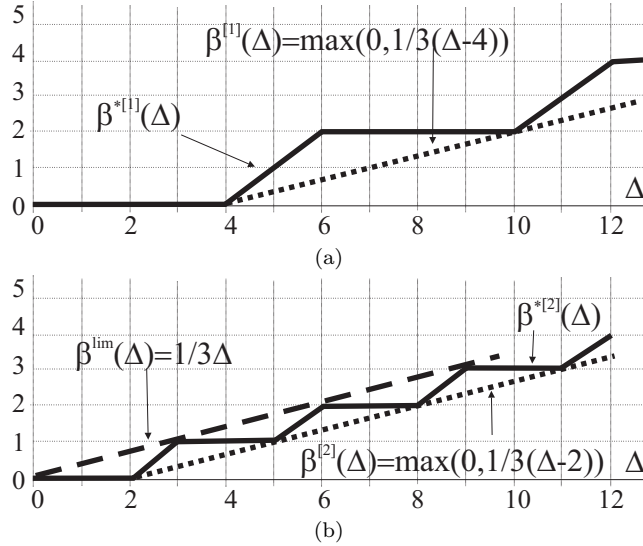
Figure 3.9: Approximating a slow processor with a unit-speed processor.

limiting service function $\beta^{lim}(\Delta) = \Delta/3$, shown in Figure 3.9(b), as the availability function.

In order to apply Theorem 3.2 to a uniform multiprocessor system, task execution times have to be measured with respect to the fastest processor. The speeds of all processors must be scaled down so that the fastest processor has unit speed. When considering a system with partially-available processors in Section 3.3, we did not make any assumptions about the way that jobs are assigned to processors except that these processors select at most m jobs of highest priority. Therefore, Corollary 3.2, under which bounded tardiness is guaranteed, may be unnecessarily restrictive for uniform multiprocessors. This is because Theorem 3.2 treats different-speed processors and partially-available unit-speed processors in a unified fashion. In the case of a uniform multiprocessor, it may be more advantageous to assign jobs with larger utilizations or execution times or higher priorities to faster processors in order to achieve better performance. Alternatively, a partitioning scheme that restricts the set of processors where jobs may execute can be employed (e.g., see (Leontyev and Anderson, 2007a)).

3.4.3 Systems With Full Processor Availability

In previous work on deriving tardiness bounds for different global scheduling algorithms (Devi and Anderson, 2005; Devi et al., 2006; Leontyev and Anderson, 2007c), a system where all processors are always available for scheduling soft real-time tasks from τ was considered. In this section, we instantiate Theorem 3.2 for this important subcase.

If all processors are fully available to tasks in τ, then for each k, $\beta_k(\Delta) = \Delta$, $\overline{F} = 0$, $\widehat{u_k} = 1$, and

$\sigma_k = 0$. Setting these values into Theorem 3.2 we have the following corollary.

Corollary 3.3. *If all processors are always available for scheduling the tasks in τ, then the tardiness of any task T_k under a window-constrained scheduling algorithm $\mathcal{A}$ is at most $\max(\rho, z) + e_k$, where*

$$z = \frac{E_L + \max(V(\ell))}{m - U_L}, \tag{3.36}$$

and $V(\ell) = -e_\ell + \alpha(\tau, \ell) + (m - 1) \cdot \rho$.

Note that the denominator of (3.36) is always positive since $U_L < m$ holds, by Definition 3.11.

3.4.4 Tightening the Bound for Specific Algorithms

The bounds in Theorem 3.2 and Corollary 3.3 can be improved for particular algorithms by exploiting the structure of the sets $\tau_{\mathbf{DH}}$ and $\tau_{\mathbf{DLH}}$, and the way jobs are prioritized. (Indeed, it is difficult to establish a tight bound when considering only very general properties of a scheduling algorithm.)

For example, for global EDF, $\chi(T_{i,j}, t) = d_{i,j}$, so jobs with deadlines after t_d have lower priority than $T_{\ell,q}$. Thus, $\phi_i = -p_i$, $\psi_i = 0$, and $\rho = 0$. By (3.12), we have $\mathbf{DH} = \emptyset$, and hence, $\mathbf{DLH} = \emptyset$, which by Definitions 3.7 and 3.13, implies $\alpha(\tau, \ell) = 0$. As a result, tardiness under global EDF for task T_k is at most

$$e_k + \max\left(0, \frac{E_L + 2\sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \max_{T_h \in \tau}\left(e_h \cdot \left(\sum_{k=1}^{m}(1 - \widehat{u_k}) - 1\right)\right)}{\sum_{k=1}^{m} \widehat{u_k} - \max(\overline{F} - 1, 0) \cdot \max(u_h) - U_L}\right), \tag{3.37}$$

provided the denominator of the second argument of max is positive.

If at most one processor is partially available, then $F \geq m - 1$, $\overline{F} \leq 1$, $\widehat{u_k} = 1$ for each k except one, and $\sigma_k = 0$ for each k except one. From this, we have

$$\left.\begin{array}{l} \sum_{k=1}^{m} \widehat{u_k} = m - 1 + \min(\widehat{u_h}), \\[4pt] \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k = \max(\widehat{u_h} \cdot \sigma_h), \\[4pt] \sum_{k=1}^{m}(1 - \widehat{u_k}) - 1 = -\min(\widehat{u_h}), \\[4pt] \max(\overline{F} - 1, 0) \cdot \max(u_h) = 0. \end{array}\right\} \tag{3.38}$$

Setting (3.38) into (3.37), we have a tardiness bound for task T_k under GEDF if at most one processor is partially available:

$$e_k + \frac{E_L + 2\max(\widehat{u_k} \cdot \sigma_k) - \min(\widehat{u_h}) \cdot \min_{T_h \in \tau}(e_h)}{m - 1 + \min(\widehat{u_h}) - U_L}. \tag{3.39}$$

Finally, if all processors are fully available, then $\max(\widehat{u_h} \cdot \sigma_h) = 0$, because $\sigma_h = 0$ for all h,

70

and $\min(\widehat{u_h}) = 1$, and hence, by (3.39), the tardiness under global EDF for task T_k is at most $e_k + \frac{E_L - \min_{T_h \in \tau}(e_h)}{m - U_L}$. The latter tardiness bound was first established by Devi and Anderson (2005).

Under global FIFO, jobs are prioritized by their release times, i.e., $\chi(T_{i,j}, t) = r_{i,j}$. We thus have $\phi_i = 0$ and $\psi_i = -p_i$ for each task T_i, and hence, by (3.9), $\rho = 0$ and $\mu = 0$. Using these values and Claim 3.2, we can upper-bound $\alpha(\tau, \ell)$ by $\sum_{T_i \in \tau \setminus T_\ell} e_i$. After setting these values into (3.36), from Corollary 3.3, the maximum tardiness of task T_k under global FIFO is $e_k + \frac{E_L + \max_\ell(\sum_{T_i \in \tau \setminus T_\ell} e_i - e_\ell)}{m - U_L}$. This bound is slightly worse than that obtained in (Leontyev and Anderson, 2007c), which is $e_k + \frac{E_L + \max_\ell(\sum_{T_i \ : \ p_i > p_\ell} e_i - e_\ell)}{m - U_L}$.

3.4.5 Non-Preemptive Execution

As shown in Section 3.2, the notion of window-constrained priorities allows a wide range of scheduling algorithms to be described. Some of these algorithms, e.g., global FIFO, execute jobs non-preemptively. Non-preemptivity is useful when overheads associated with rescheduling are high or when exclusive access to shared resources is needed. Some simple but efficient resource access protocols require using short non-preemptive code regions (Block et al., 2007).

Non-preemptive execution causes priority inversions when a lower-priority job is scheduled and a higher-priority job is ready but not scheduled. In this section, we show how to model non-preemptivity using window-constrained prioritization functions in a system where all processors are always available for scheduling the tasks in τ; we leave the analysis of non-preemptive execution under partial processor availability as an open problem. (Indeed, it is not clear how to deal with the situation where a processor becomes unavailable while a job is executing on it non-preemptively.) We assume some additional constraints on the task system and the scheduler.

Definition 3.17. We call a task system *restricted early-release* if there exists a constant $\gamma \geq 0$ such that, for each job $T_{i,j}$,

$$\epsilon_{i,j} \geq r_{i,j} - \gamma. \tag{3.40}$$

Definition 3.18. Let $\chi^A(T_{i,j}, t)$ be a prioritization function imposed by the scheduling algorithm $\mathcal{A}$. We call $\mathcal{A}$ *eventually-monotonic* if there exists a constant $M \geq 0$ such that for each job $T_{i,j}$, for all $t \geq d_{i,j} + M$ and $v \geq 0$, $\chi^A(T_{i,j}, t) \leq \chi^A(T_{i,j}, t + v)$.

From the above definition, any algorithm for which $\chi^A(T_{i,j}, t)$ is constant, e.g., global EDF, FIFO, and RM, is eventually-monotonic. Also, it is easy to verify that LLF and EDZL, as specified as in Examples 3.5 and 3.6, are eventually-monotonic. In the rest of this section, we concentrate on restricted early-release task systems scheduled under an eventually-monotonic scheduler $\mathcal{A}$ assuming that (3.3)

holds for $\chi^A(T_{i,j}, t)$. We show how to modify the prioritization functions of $\mathcal{A}$ in a window-constrained way to ensure non-preemptive execution (if this is not ensured already).

Definition 3.19. Let $\phi_{max} = \max_{T_i \in \tau}(\phi_i)$, $p_{max} = \max_{T_i \in \tau}(p_i)$, and $G = \mu + \gamma + \phi_{max} + M + p_{max} + 1$.

As mentioned earlier, non-preemptive execution causes priority inversions when a low-priority job $T_{i,j}$ is scheduled and there is a ready high-priority job $T_{a,b}$ that is not scheduled. This means that $T_{i,j}$'s priority is effectively higher than that of $T_{a,b}$ for the duration of the non-preemptive region. We can explicitly model this behavior by changing prioritization functions of $\mathcal{A}$ as follows.

If a ready job $T_{i,j}$ is not executing within a non-preemptive region, then $\chi(T_{i,j}, t) = \chi^A(T_{i,j}, t)$. If $T_{i,j}$ begins executing a non-preemptive region at time t_1 and leaves that region at a later time t_2, then we "boost" its priority while it executes non-preemptively by setting $\chi(T_{i,j}, t) = r_{i,j} - G$ for all $t \in (t_1, t_2)$.

Theorem 3.3. (proved in the appendix) *If $\mathcal{A}$ is an eventually-monotonic scheduling algorithm and its prioritization functions are augmented as described above, then no job is preempted while executing in a non-preemptive region.*

The augmented prioritization function $\chi(T_{i,j}, t)$ remains window-constrained because $r_{i,j} - G \leq \chi(T_{i,j}, t) \leq d_{i,j} - \psi_i$ holds, where G is constant. By Corollary 3.3, this implies that tardiness is bounded for any restricted early-release task system under a window-constrained eventually-monotonic scheduler on m fully available processors even if the tasks in τ have non-preemptive regions.

3.5   Experiments

As noted in Section 3.4, different algorithms to which Theorem 3.2 applies may exhibit very different behavior in terms of tardiness. To provide a sense of how significant such differences can be, we present here the results of some experiments that we conducted to compare observed tardiness under different scheduling algorithms.

In these experiments, we examined m-procssesor systems for which task sets were randomly generated. Each task in such a task set was generated by selecting an integral execution time, uniformly distributed over the range $[1, 10]$, and a utilization, uniformly distributed over the range $[u_{min}, u_{max})$. We considered three utilization ranges: $[0.01, 0.05)$ (light), $[0.05, 0.5)$ (medium), and $[0.5, 0.9)$ (heavy). For each utilization range, a seed task set τ of total utilization at least $(m + 1)/2$ was generated, and

then additional task sets were successively generated by adding tasks to τ until total utilization exceeded m. This process was then repeated until a total of 500 seed task sets had been generated (for that utilization range). For each resulting task set, we produced schedules (with job releases occurring in a synchronous, periodic manner) for each of EDF, FIFO, LLF, and EDZL for $\min(20000, 20 \cdot \max(p_i))$ time units. The selected interval lengths are not guaranteed to be larger than the least common multiple of the periods of the tasks in each generated task set. However, for a subset of the generated task sets, we simulated significantly longer schedules and found that tardiness did not grow significantly beyond the $\min(20000, 20 \cdot \max(p_i))$ threshold. In producing schedules, system and scheduling overheads were taken to be negligible. For each schedule, the maximum observed tardiness was recorded.

Figure 3.10 shows the maximum observed tardiness values under EDF, FIFO, LLF, and EDZL as a function of U_{sum} for $m = 4$ for the light (inset (a)), medium (inset (b)), and heavy (inset (c)) utilization ranges. These observed values are denoted O-GEDF, O-FIFO, O-LLF, and O-EDZL, respectively. Additionally, for each task set, a maximum tardiness bound under LLF and EDZL was computed using Corollary 3.3 and assuming $\psi_i = \phi_i = 0$ for each task T_i. This bound is denoted C-GEN (it is a generalized bound, which is also applicable to FIFO and EDF). We also computed tighter bounds for EDF and FIFO, denoted C-GEDF and C-FIFO, respectively, as discussed in Section 3.4.4. To compute the maximum deadline tardiness under FIFO, we used the slightly improved bound mentioned earlier in Section 3.4.4 from (Leontyev and Anderson, 2007c). Figure 3.11 depicts similar data for the case $m = 8$.

Of the four scheduling algorithms under consideration, observed tardiness under LLF and EDZL was smaller than that under FIFO and EDF (*much* smaller than under FIFO). While LLF may be impractical in reality because it preempts jobs frequently, EDZL could be a viable approach for scheduling soft real-time workloads when tardiness is allowed.

The general tardiness bound obtained using Corollary 3.3 is five to six times larger than the maximum task execution time, which seems quite reasonable, for the medium and heavy per-task utilization ranges (see insets (b) and (c) of Figures 3.10 and 3.11). In contrast, for the light utilization range, the maximum tardiness bound is about twenty times larger than the maximum per-task execution cost. However, the observed tardiness under FIFO for that utilization range is also quite high so it is unlikely that the general bound can be improved much (see inset (a) of Figures 3.10 and 3.11). Even though observed tardiness under LLF and EDZL is practically zero, the tardiness bound given for them by Corollary 3.3 (C-GEN) is very pessimistic, due to the use of a conservative estimation for $\alpha(\tau, \ell)$ (from Claim 3.2). Obtaining a better estimation for these algorithms is difficult, due to their dynamic nature.

The experiments also show that the FIFO bound improvement discussed in Section 3.4.4 is only a
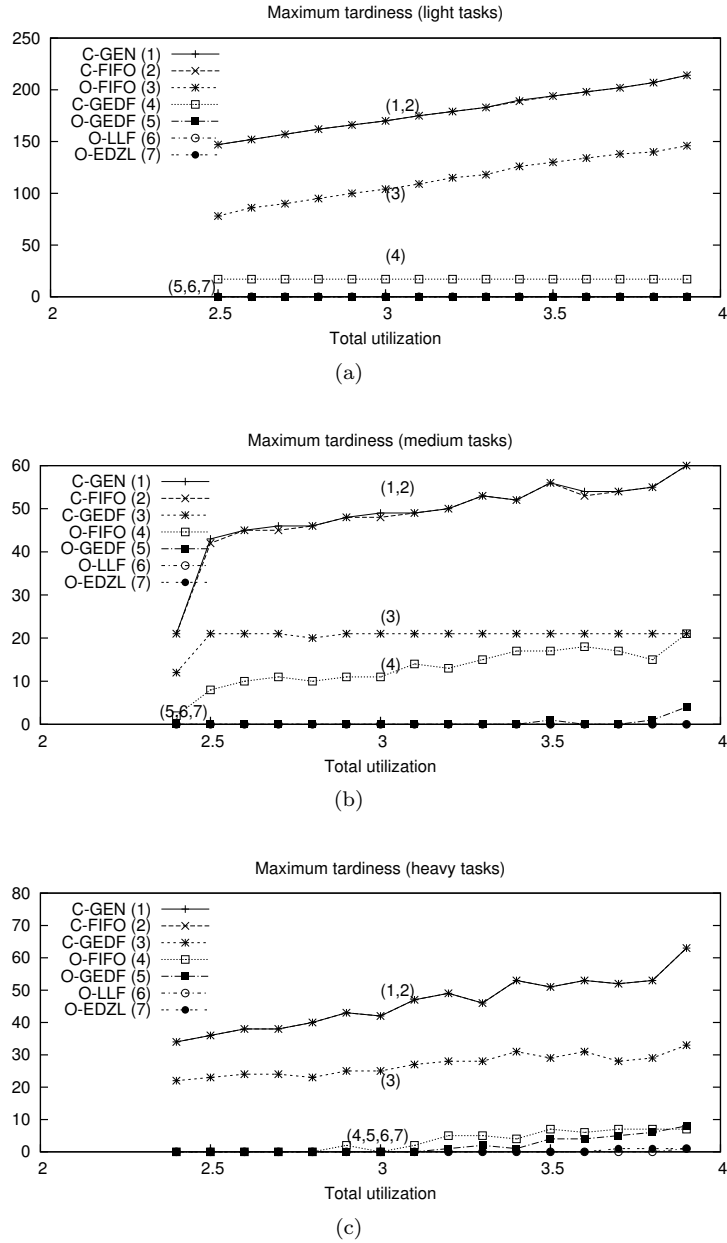
Figure 3.10: Maximum deadline tardiness observed and computed for **(a)** light, **(b)** medium, and **(c)** heavy per-task utilization ranges for $m = 4$ processors.

slight improvement (C-GEN and C-FIFO do not differ much in any graph). In contrast, the improved bound for EDF is significantly better. (Note that the improved bound for EDF is two to three times larger than the maximum per-task execution time for all utilization ranges.) These results suggest that it might be possible to improve the tardiness bound for each algorithm (particularly EDZL and LLF) further. We leave the development of tighter bounds for these algorithms as open problems.
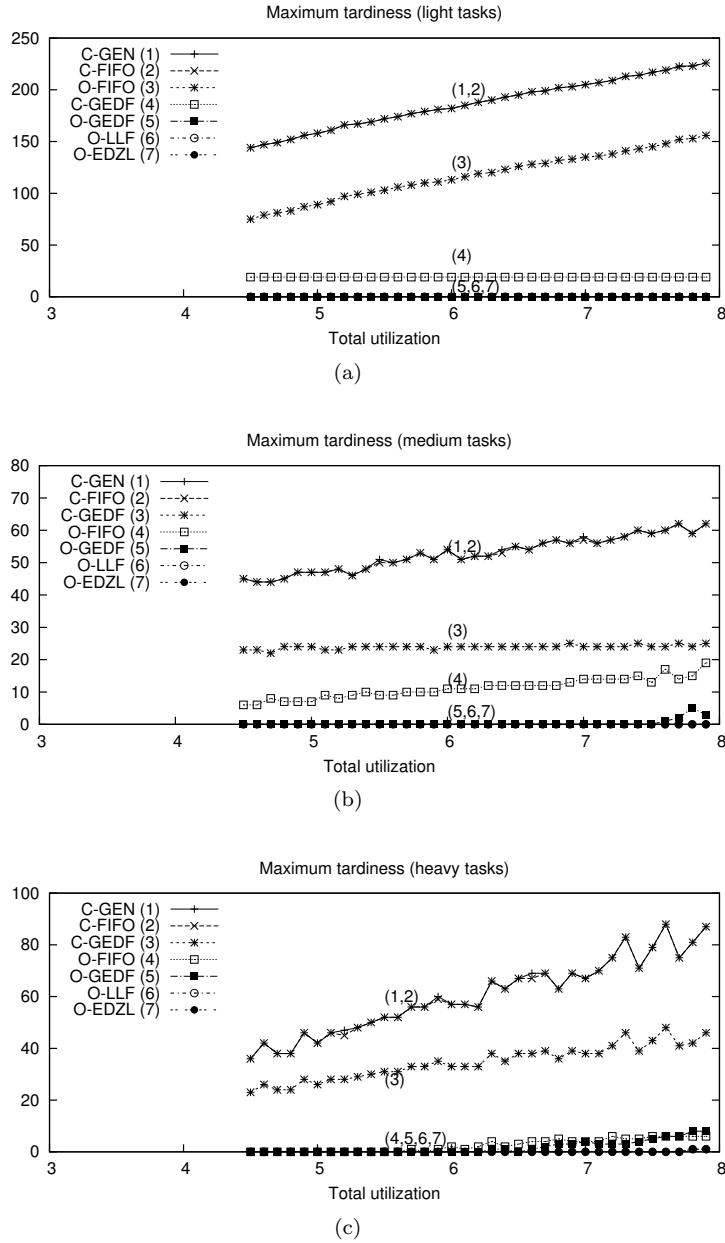
Figure 3.11: Maximum deadline tardiness observed and computed for **(a)** light, **(b)** medium, and **(c)** heavy per-task utilization ranges for $m = 8$ processors.

3.6 Summary

In this chapter, we have presented a general tardiness-bound derivation that applies to a wide variety of global scheduling algorithms. Our results show that, with the exception of static-priority algorithms, most global algorithms of interest in the real-time-systems community have bounded tardiness. When considering new algorithms, the question of whether tardiness is bounded can be answered in the affirmative by simply showing that the required prioritization can be specified. Of course, a tardiness bound that is tighter than that given by our results might be possible through the use of reasoning specific

to a particular algorithm. Indeed, it is difficult to obtain a very tight bound when assuming so little concerning the nature of the scheduling algorithm. Our goal in this chapter was not to produce the tightest bound possible, but rather to produce a bound that could be widely applied. We leave as an open question whether the existence of a window-constrained prioritization for a scheduling algorithm is a *necessary* condition for bounded tardiness.

Several interesting avenues for further work exist. First, it would be interesting to investigate reactive techniques that could be applied at runtime to lessen tardiness for certain jobs by redefining priority points, as circumstances warrant. Such techniques might exploit the fact that our framework allows priority definitions to be changed rather arbitrarily at runtime. Second, our experimental results suggest that actual tardiness under EDZL is likely to be very low. It would be interesting to improve our analysis as it applies to EDZL in order to obtain a tight tardiness bound.

Chapter 4

A Hierarchical Bandwidth Reservation

Scheme with Timing Guarantees

Using the results from the previous chapter, in this chapter[1], we design a multiprocessor scheduling scheme for supporting hierarchical containers that encapsulate sporadic soft and hard real-time tasks.

The rest of this chapter is organized as follows. In Section 4.1, we present our container model. In Section 4.2, we formally characterize the "supply" available to a container and propose a container scheduling scheme. In Sections 4.3 and 4.4, we present methods for checking the schedulability of real-time tasks within a container and for computing the supply available to its child containers (if any). In Section 4.5, we discuss tradeoffs pertaining to having hard real-time tasks in containers. In Section 4.6, we examine the extent to which temporal isolation is ensured in container hierarchies under our scheduling scheme. In Section 4.7, we present our experimental results. We conclude the chapter in Section 4.8.

4.1   Container Model

In order to support the scheduling of containers within an arbitrary hierarchy, it suffices to consider the problem of scheduling a single container H on a set of $M(H)$ unit-speed processors, where some processors may not be available for execution during certain time intervals. The set of child containers and real-time tasks encapsulated in H is referred to as $succ(H)$. (Non-real-time tasks could be contained

---

[1]Contents of this chapter appeared previously in the following papers:

Leontyev, H. and Anderson, J. (2008a). A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems*, pages 191–200.

Leontyev, H. and Anderson, J. (2009b). A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. *Real-Time Systems*, 43(1):60–92.
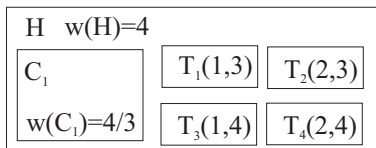
Figure 4.1: A host container H that encapsulates another container C_1 and four real-time tasks $T_1, \ldots, T_4$.

as well, but we do not consider such tasks in this dissertation.) At any time, the container may be scheduled on several available processors. When the container is scheduled, some of its children are selected for execution using some internal scheduling policy.

The set of implicit-deadline tasks encapsulated in the container H is denoted $\tau = \{T_1, \ldots, T_n\}$. In that which follows, we find it convenient to view a real-time task as a specialized container with no nested children that can be scheduled on at most one processor at any time and that has hard or soft deadlines.

Container bandwidth. Each container H is characterized by its *bandwidth* $w(H) \geq 0$, which specifies the processing capacity to which it is entitled. For a real-time task T_i, we define $w(T_i) \triangleq u_i$. Since the containers in $succ(H)$ are scheduled when the parent container is scheduled, their allocation time cannot exceed that of H. Therefore, we require

$$w(H) \geq \sum_{C_j \in succ(H)} w(C_j). \tag{4.1}$$

Example 4.1. In Figure 4.1, a host container H with bandwidth $w(H) = 4$ encapsulates a child container C_1 with bandwidth $w(C_1) = 4/3$, two HRT tasks $T_1(1,3)$ and $T_2(2,3)$, and two SRT tasks $T_3(1,4)$ and $T_4(2,4)$.

Overview of our approach. In the following sections, we solve the problem described at the beginning of this section via a decomposition into two subproblems, each of which can be solved by applying previously-published results. First, we split the bandwidth of each container, parent and child, into integral and fractional parts and argue that the integral parts can easily be dealt with. The fractional part of each child container is then handled by creating a special SRT *server task* with utilization equal to that fractional portion. This leads to our first subproblem, which is that of scheduling within the parent container, using the "supply" available to it, all child HRT and SRT tasks (where some of the SRT tasks may be server tasks). We then deal with any HRT tasks by encapsulating them within a new child container that schedules these tasks on an integral number of processors via a prior HRT scheduling

78

scheme. This leaves us with our second subproblem, which is to schedule within the parent container a collection of SRT tasks. We solve this problem by exploiting the fact that window-constrained global scheduling algorithms ensure bounded tardiness, as shown in Chapter 3. So that our overall scheme can be applied recursively in a container hierarchy, we finish our analysis by characterizing the supply available to each child container.

4.2 Container Scheduling

The host container H receives processor time from $M(H)$ individual processors. We now further constrain the manner in which any container C receives processor time by assuming the following.

(P) At any time, a container C can be scheduled on $m(C) \triangleq \lfloor w(C) \rfloor$ or $M(C) \triangleq \lceil w(C) \rceil$ processors.

This restriction minimizes the execution parallelism available to C so that, for any interval of length Δ, C's allocation is within $[\lfloor w(C) \rfloor \Delta, \lceil w(C) \rceil \Delta]$. For real-time tasks, this restriction holds implicitly, because a real-time task T_i is scheduled on at most one processor at any time and $w(T_i) = u_i \leq 1$, so $\lceil w(T_i) \rceil = 1$ and $\lfloor w(T_i) \rfloor = 0$. We say that a processor is *fully available* to C, if it is dedicated exclusively to C. Given Restriction (P), we can assume that $m(C)$ processors are fully available to C.

As explained in detail later, there are two reasons for introducing Restriction (P). First, increasing the amount of supply parallelism (the number of available processors) restricts the maximum per-task utilization and the total system utilization if the long-term supply remains fixed. Second, maximizing the number of processors fully available to C lessens deadline tardiness for any child real-time task. Intuitively, this is because such tasks are *sequential* and thus may leave processors unused if parallelism is increased too much.

Example 4.2. Consider a container H with bandwidth $w(H) = 4/3$ that encapsulates a task $T_1(5, 6)$, as shown in Figure 4.2(a). Suppose that processor time is supplied as shown in Figure 4.2(b) so that H occupies two processors for two time units every three time units. The supply available to H is approximately $\frac{4 \cdot \Delta}{3}$ for any sufficiently long interval Δ. However, H does not execute during the interval $[2, 3)$, so Restriction (P) is violated, because $\lfloor w(H) \rfloor = \lfloor 4/3 \rfloor = 1$. Task T_1's jobs demand five execution units every six time units, but because they must execute sequentially, they can execute for only four time units every six time units. Thus, task T_1's tardiness can be unbounded. In the schedule in Figure 4.2(c), container H also receives four execution units every three time units, but in contrast to Figure 4.2(b), Restriction (P) is satisfied. Because one processor is fully available to H, task T_1 meets all of its deadlines.
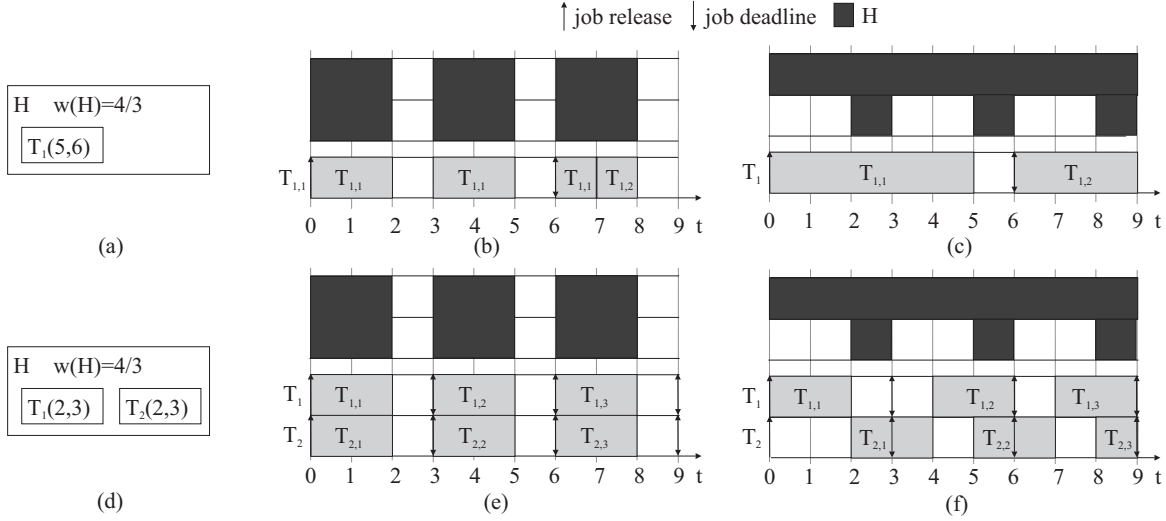
Figure 4.2: Comparison of supply parallelism in Examples 4.2 and 4.3.

As one may suspect, enforcing Restriction (P) may sometimes have negative consequences. Indeed, a task set with a large number of tasks may benefit from a larger number of available processors if all deadlines have to be met.

Example 4.3. Consider the container H from the previous example, except that it now encapsulates two real-time tasks $T_1(2,3)$ and $T_2(2,3)$, as shown in Figure 4.2(d). In the schedule shown in Figure 4.2(e), which is equivalent from the container's perspective to that in Figure 4.2(b), jobs of T_1 and T_2 meet their deadlines. However, in the schedule in Figure 4.2(f), where Restriction (P) is enforced as in Figure 4.2(c), job $T_{2,1}$ misses its deadline at time 3 because it cannot execute on two processors simultaneously during the time interval $[2,3)$. Still, in this schedule, T_2's tardiness is only one time unit.

The two examples above illustrate that, while minimizing supply parallelism may negatively impact timeliness, it allows the widest range of loads to be scheduled with bounded deadline tardiness, which is in accordance with our focus on SRT tasks.

We now develop a scheduling policy that enforces Restriction (P) for child containers assuming that it holds for the host container H. Given the latter, H is supplied time from $M(H)$ processors, where $m(H)$ processors are always available for scheduling $succ(H)$ and at most one processor is partially available.

A child container $C_i \in succ(H)$ must occupy at least $m(C_i)$ processors at any time. By (4.1), $w(H) \geq \sum_{C_i \in succ(H)} w(C_i)$, and hence, $m(H) = \lfloor w(H) \rfloor \geq \lfloor \sum_{C_i \in succ(H)} w(C_i) \rfloor \geq \sum_{C_i \in succ(H)} \lfloor w(C_i) \rfloor$ $= \sum_{C_i \in succ(H)} m(C_i)$. Therefore, we can make $m(C_i)$ processors fully available to each child container $C_i \in succ(H)$ by using the $m(H)$ processors fully available to H. Note that, for containers with $w(C_i) < 1$

80

(including real-time tasks), $m(C_i) = \lfloor w(C_i) \rfloor = 0$. In any event, given this design decision, each child container C_i receives at least $m(C_i)\Delta$ units of time over an interval of length Δ.

If a child container C_i is not a real-time task and $m(C_i) < w(C_i)$, then it occasionally needs supply from an additional processor. For this, we construct a SRT periodic *server task* $S_i(e_i, p_i)$, where $u_i = e_i/p_i = w(C_i) - m(C_i) < 1$. (The term *periodic* means that $r_{i,j} = (j-1) \cdot p_i$ holds for each $j \geq 1$.)

We denote the set of server tasks as $\tau^S = \{S_1, \ldots, S_n\}$. Jobs of these tasks are scheduled together with the jobs of encapsulated real-time tasks using the remaining $m(H) - \sum_{C_j \in succ(H)} m(C_j)$ fully available processors and at most one partially available processor. When task S_i's jobs are scheduled, an additional processor is available to container C_i. Because server task S_i is constructed only if $w(C_i) > m(C_i) = \lfloor w(C_i) \rfloor$, we have $\lceil w(C_i) \rceil = m(C_i) + 1 = M(C_i)$. Thus, container C_i always occupies $m(C_i)$ processors, and $M(C_i)$ processors are occupied when a job of S_i is scheduled. Thus, Restriction (P) is ensured for each child container.

Example 4.4. Consider container H from Example 4.1. For container C_1, one processor is reserved because $\lfloor w(C_1) \rfloor = \lfloor 4/3 \rfloor = 1$. For this container, we also construct a SRT server task $S_1(1, 3)$, so that $\lfloor w(C_1) \rfloor + e_1/p_1 = 1 + 1/3 = w(C_1)$. When jobs of S_1 are scheduled, an additional processor is available to container C_1, as shown in Figure 4.3(b).

Let $\mathsf{HRT}(H)$ (respectively, $\mathsf{SRT}(H)$) be the set of HRT (respectively, SRT) tasks encapsulated in H. The remaining problem at hand, referred to as Subproblem 1, is that of scheduling tasks from the sets $\mathsf{HRT}(H)$, $\mathsf{SRT}(H)$, and τ^S on some number of fully available processors and at most one partially available processor.

4.3   Subproblem 1

To schedule the tasks in $\mathsf{HRT}(H)$, we encapsulate them into a child container C_{hrt} with integral bandwidth $w(C_{hrt}) = m(C_{hrt}) = M(C_{hrt})$. Applying Restriction (P) to C_{hrt}, $m(C_{hrt})$ processors must be reserved for this container. In this section, we consider two approaches for scheduling the remaining tasks in $\mathsf{SRT}(H)$ and τ^S; in the first approach, HRT and SRT tasks do not execute on the same processors, and in the second approach, they may.

Basic approach.   The tasks in $\mathsf{HRT}(H)$ can be scheduled within C_{hrt} using a variety of approaches. Given our emphasis on SRT tasks, we simply use the partitioned EDF (PEDF) algorithm for this purpose, deferring consideration of other approaches to future work. Under PEDF, tasks are statically assigned
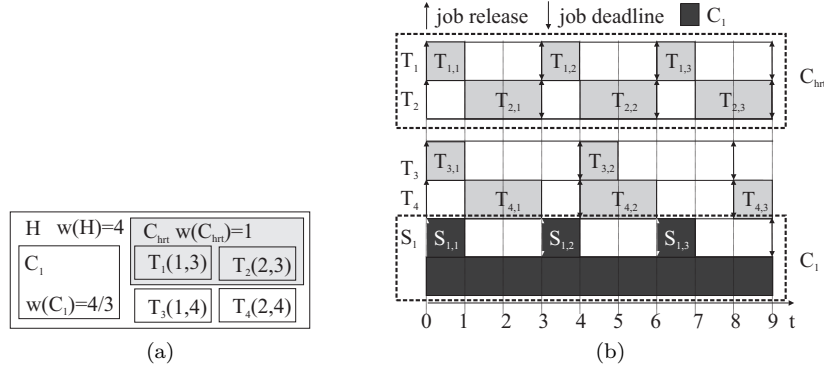
Figure 4.3: Example 4.5. **(a)** Isolating HRT tasks. **(b)** A schedule with the two HRT tasks in a separate container.

to processors and each processor schedules its assigned tasks independently on an EDF basis. Assume that processor h is among the $m(C_{hrt})$ processors reserved for container C_{hrt} and let τ_h denote the set of sporadic HRT tasks assigned to that processor. All task deadlines will be met on processor h if

$$U_{sum}(\tau_h) = \sum_{T_i \in \tau_h} u_i \leq 1, \tag{4.2}$$

which is a well-known uniprocessor EDF schedulability test (Liu and Layland, 1973). This test, when applied in a multiprocessor system, presumes a given assignment of tasks to processors. Such an assignment (and correspondingly, the number of processors required for C_{hrt}) can be determined using any of various bin-packing heuristics. Further results concerning PEDF schedulability tests can be found in (Baruah and Fisher, 2006, 2007; Chakraborty and Thiele, 2005; Liu, 2000).

As mentioned earlier in Sections 1.5.2 and 2.1.1, HRT policies may introduce utilization loss. For PEDF, there exist task sets, for which the reserved processors could be underutilized. However, if HRT tasks are relatively few in number, such loss will likely be small, compared to the total utilization of SRT tasks. Loss is incurred when creating C_{hrt} if its bandwidth (given by the number of processors required for it) exceeds the sum of the utilizations of the HRT tasks it contains. If this is the case, then (4.1) must be validated with the tasks in HRT(H) replaced by the container C_{hrt}.

Example 4.5. Consider again container H from Example 4.1. In our approach, we encapsulate the two HRT tasks $T_1(1,3)$ and $T_2(2,3)$ into a container C_{hrt}, as shown in Figure 4.3(a). The total utilization of these two tasks is $U_{sum} = u_1 + u_2 = 1/3 + 2/3 = 1$. By (4.2), these two tasks will meet their deadlines if scheduled using uniprocessor EDF. We set $w(C_{hrt}) = 1$, so the container C_{hrt} will require one processor. The total bandwidth of container H's children is $\sum_{C_i \in succ(H)} w(C_i) = w(C_1) + w(C_{hrt}) + w(T_3) + w(T_4) =$

82

$4/3 + 1 + 1/4 + 2/4 = 37/12 < 4 = w(H)$, so (4.1) is satisfied. When scheduling the modified container H on $\lceil w(H) \rceil = 4$ processors, as shown in Figure 4.3(b), one processor is reserved for the HRT container C_{hrt} and tasks T_1 and T_2 are scheduled on that processor. Note that no utilization loss is incurred by HRT tasks. In Example 4.4, we reserved one processor for container C_1 and constructed the server task $S_1(1, 3)$. Jobs of this server task are scheduled with the jobs of tasks T_3 and T_4 on the two remaining fully available processors.

Note that, if a system has a small number of processors, then it may not be possible to dedicate an integral number of processors for a HRT container as described above. For example, if the parent container H has fractional bandwidth, then its encapsulated HRT tasks may be required to execute on a partially available processor. In this case, the HRT schedulability of these tasks can be checked using a test such as that described in Section 2.3.3. However, if a system is purely SRT, an arbitrarily deep hierarchy of SRT containers can be maintained even in the uniprocessor case.

In the case when it is possible to reserve an integral number of processors for HRT tasks, it may not be possible to accommodate SRT tasks using the remaining bandwidth as the following example illustrates.

Example 4.6. Consider Figure 4.4(a), which depicts a container H that is similar to that from Example 4.1, except that T_2 has a smaller execution time and there are two additional SRT tasks, $T_5(1, 2)$ and $T_6(1, 2)$. In our approach, we encapsulate the two HRT tasks $T_1(1, 3)$ and $T_2(1, 3)$ into a container C_{hrt}, as shown in Figure 4.4(a). The total utilization of these two tasks is $U_{sum} = u_1 + u_2 = 1/3 + 1/3 = 2/3$. By (4.2), these two tasks will meet their deadlines if scheduled using uniprocessor EDF. We set $w(C_{hrt}) = 1$, so the container C_{hrt} requires one processor. When scheduling the modified container H on $\lceil w(H) \rceil = 4$ processors, as shown in Figure 4.4(b), one processor is reserved for the HRT container C_{hrt}, and tasks T_1 and T_2 are scheduled on that processor (inset (c) is considered later). As in Example 4.4, we reserve one processor for container C_1 and construct a server task $S_1(1, 3)$. Jobs of this server task are scheduled with the jobs of tasks $T_3, \ldots, T_6$.

Under the basic approach, the processor time that remains after scheduling T_1 and T_2 is unused (see intervals $[2, 3)$ and $[5, 6)$ within C_{hrt} in Figure 4.4(b)). Thus, the bandwidth available to tasks S_1 and $T_3, \ldots, T_6$ is $w(H) - m(C_{hrt}) - m(C_1) = 4 - 1 - 1 = 2$. However, the total bandwidth required by tasks S_1 and $T_3, \ldots, T_6$ is $w(S_1) + w(T_3) + w(T_4) + w(T_5) + w(T_6) = 1/3 + 1/4 + 2/4 + 1/2 + 1/2 = 25/12 > 2$, and hence, tasks S_1 and $T_3, \ldots, T_6$ will have unbounded deadline tardiness. Note that, in the schedule in Figure 4.4(b), the ready job $T_{6,2}$ is not scheduled during the interval $[2, 3)$ even though there is an
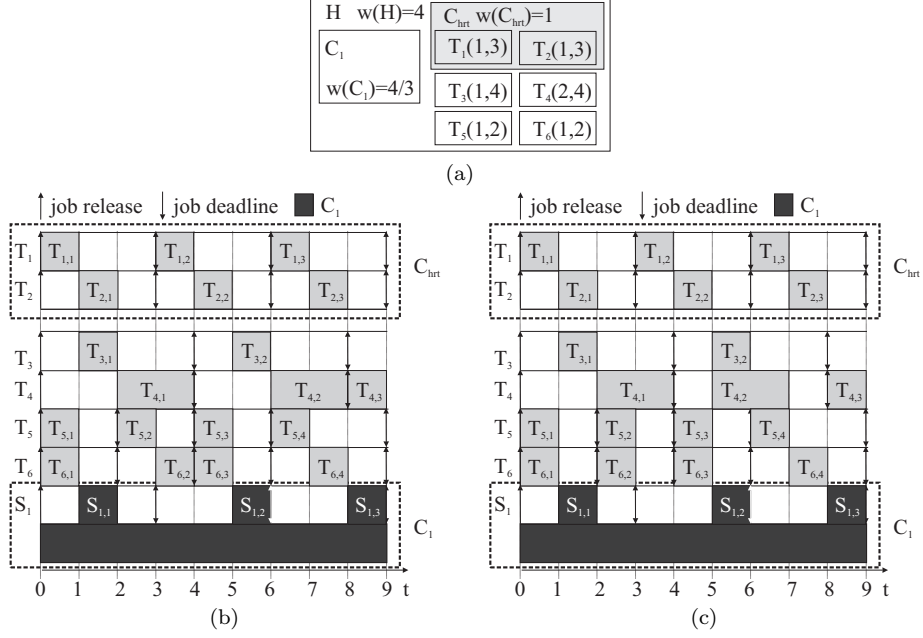
Figure 4.4: **(a)** Container considered in Examples 4.6 and 4.7. A schedule **(b)** with and **(c)** without HRT time reclamation.

available processor. Similarly, the ready job $T_{4,2}$ is not scheduled during the interval $[5, 6)$.

Extended approach. In order to allocate the available bandwidth more efficiently, we can use the time not allocated to HRT tasks on some of the $m(C_{hrt})$ processors reserved for such tasks to schedule tasks in $\mathsf{SRT}(H) \cup \tau^S$ (in addition to the supplied time on other processors). We allow this approach to be selectively applied by defining the parameter $K(H)$ below.

Definition 4.1. Let $K(H) \in [0, m(C_{hrt})]$ be the number of processors where tasks in $\mathsf{HRT}(H)$ and $\mathsf{SRT}(H) \cup \tau^S$ are co-scheduled.

We assume that HRT tasks are statically prioritized over SRT and server tasks. Thus, HRT tasks still execute as if an integral number of processors were dedicated to their exclusive use. After assigning all HRT tasks to the $m(C_{hrt})$ processors reserved for them and then selecting $K(H)$, the utilization loss due to partitioning is $U_{\mathsf{lost}} = \sum_{k=K(H)+1}^{m(C_{hrt})} (1 - U_{sum}(\tau_k))$ (we assume that HRT-allocated processors are numbered in order of increasing $U_{sum}(\tau_k)$). Though engaging additional processors for scheduling tasks in $\mathsf{SRT}(H) \cup \tau^S$ (i.e., increasing $K(H)$) reduces utilization loss and sometimes is imperative in order to accommodate all SRT tasks, a large value for $K(H)$ may negatively impact SRT schedulability as discussed later in Section 4.4; tradeoffs involved in selecting $K(H)$ are discussed in Section 4.5. After weighing such tradeoffs and selecting a value for $K(H)$, (4.3) below must be validated to account for

84

any lost bandwidth.

$$w(H) \geq \sum_{C_j \in succ(H)} w(C_j) + U_{\text{lost}} \tag{4.3}$$

Example 4.7. Consider container H from Example 4.6. A schedule where HRT processor time is reclaimed (i.e., $K(H) = 1$) is shown in Figure 4.4(c). The bandwidth available to tasks S_1 and $T_3, \ldots, T_6$ is $w(H) - w(T_1) - w(T_2) - m(C_1) = 4 - 1/3 - 1/3 - 1 = 7/3$, which is greater than the bandwidth required by these tasks. Note that, in this schedule, the processors supplied to H are idle only if there are not enough ready tasks to occupy all of them.

Having dispensed with any HRT tasks, we can complete our solution to Subproblem 1 by devising a scheduling policy that ensures bounded tardiness for the remaining SRT tasks, some of which may be server tasks.

Definition 4.2. (τ_s, M_s, and Subproblem 2) Let $\tau_s = \mathsf{SRT}(H) \cup \tau^S$. These tasks are to be scheduled on M_s processors, of which $m(H) - \sum_{C_j \in succ(H)} m(C_j) - m(C_{hrt})$ are fully available and $K(H) + G$, where $G \leq 1$, are partially available. Note that $K(H)$ processors are partially available due to HRT tasks internal to H and at most one additional processor is partially available because the supply provided by H's parent is subject to Restriction (P).

We refer to this last remaining subproblem as Subproblem 2.

4.4 Subproblem 2

In solving Subproblem 2, restrictions on supplied processor time are of relevance. From Definition 4.2, of the M_s processors under consideration, $K(H) + G$ are partially available. We assume that these M_s processors are indexed so that the supply from them can be described using M_s supply functions: $\beta_k^l(\Delta) = \max(0, \widehat{u_k}(\Delta - \sigma_k))$, where $0 < \widehat{u_k} \leq 1$ and $\sigma_k \geq 0$, for $1 \leq k \leq K(H) + G$; and $\beta_k^l(\Delta) = \Delta$, for $K(H) + G + 1 \leq k \leq M_s$. If $K(H) + G \leq 1$, i.e., at most one processor is partially available, then we say that such a collection of functions is in *Minimum Parallelism* (MP) form. As explained later, ensuring that supply is in MP form allows the widest range of SRT workloads to be supported without incurring utilization loss.

Before continuing, note that if $M_s = 1$, i.e., all remaining SRT tasks are to be scheduled on one processor, then EDF can be used on that processor. If this processor is fully available, then tardiness will be zero for these tasks (due to the optimality of EDF), and if it is partially available, then it can be easily shown to be bounded, using real-time calculus (Chakraborty and Thiele, 2005), provided

$U_{sum}(\tau_s) \leq \widehat{u_1}$. In the remainder of this section, we concentrate on the more interesting case, $M_s \geq 2$. In this case, our approach leverages the results from Chapter 3. We next briefly remind the reader about the relevant problem setup.

Let τ be a set of implicit-deadline SRT tasks scheduled on $M \geq 2$ processors, with supply functions $\beta_k^l(\Delta) = \max(0, \widehat{u_k}(\Delta - \sigma_k))$, where $1 \leq k \leq M$. (Note that τ was defined earlier in Section 1.3. Here, we mean τ to denote any sporadic SRT task set. The distinction should be clear from the context.) Assume

$$U_{sum}(\tau) \leq \sum_{k=1}^{M} \widehat{u_k}, \tag{4.4}$$

i.e., the total system utilization is at most the total supplied bandwidth. Released jobs are placed into a single global ready queue. When choosing a new job to schedule, the scheduler selects (and dequeues) the ready job of highest priority. Job priorities are determined as defined in Definition 3.1 (see Section 3.3). We assume that the scheduling algorithm's prioritization function is window-constrained as defined in Definition 3.4 (see Section 3.3). Below, we repeat a definition that will be often referred in the rest of this chapter.

Definition 3.11 (see Section 3.3). $(U(\tau, y)$ **and** $E(\tau, y))$ Let $U(\tau, y)$ $(E(\tau, y))$ be the set of at most $\min(|\tau|, y)$ tasks from τ of highest utilization (execution cost), where $|\tau|$ is the number of tasks in τ, and let

$$E_L = \sum_{T_i \in E(\tau, m-1)} e_i \quad \text{and} \tag{4.5}$$

$$U_L = \sum_{T_i \in U(\tau, m-1)} u_i. \tag{4.6}$$

4.4.1 Minimizing the Tardiness Bound

In Section 3.3, we established Theorem 3.2, which gives maximum tardiness bounds for implicit-deadline SRT task systems subject to (4.4) scheduled on a restricted-capacity platform. Given this theorem, we now argue in favor of Restriction (P) and show how enforcing this restriction affects the tardiness bound in Theorem 3.2. Consider the denominator of (3.33) (see Section 3.3 in Chapter 3; note that $m = M$):

$$\sum_{k=1}^{M} \widehat{u_k} - \max(\overline{F} - 1, 0) \cdot \max_{1 \leq \ell \leq |\tau|} (u_\ell) - U_L. \tag{4.7}$$

The requirement for (4.7) to be positive implicitly restricts the maximum per-task utilization if $\overline{F} > 1$, i.e., if two or more processors are partially available. Note also that the value of x is minimized if (4.7) is maximized. Suppose that the total supplied bandwidth $W = \sum_{k=1}^{M} \widehat{u_k}$ is fixed. Then, (4.7) will be maximized if either $\max(\overline{F} - 1, 0) \cdot \max_{1 \leq \ell \leq |\tau|}(u_\ell)$ or U_L or both are minimized. The value of U_L depends exclusively on task utilizations and the total number of processors M, as (4.6) suggests. Therefore, U_L will be minimized if the total number of processors M is minimized. The expression $\max(\overline{F} - 1, 0) \cdot \max_{1 \leq \ell \leq |\tau|}(u_\ell)$ is minimized if $\overline{F} \leq 1$, that is, at most one processor is partially available. Thus, if the total processor bandwidth W is fixed, then (4.7) is maximized by setting $M = \lceil W \rceil$ and having $\lfloor W \rfloor$ processors fully available. The bandwidth of at most one partially available processor (if any) is $\widehat{u_1} = W - \lfloor W \rfloor$.

The above discussion suggests that bounded tardiness among SRT and server tasks can be achieved for the widest range of task utilizations if the supply to $\mathsf{SRT}(H) \cup \tau^S$ is given in MP form. This is the case if either $K(H) = 0$, (e.g., when $\mathsf{HRT}(H) = \emptyset$ or no spare HRT capacity is reused) or $G = 0$ and $K(H) \leq 1$ (i.e., when the bandwidth supplied to H is integral and HRT capacity is reused on at most one processor). If $K(H) + G > 1$, then bounded tardiness may be guaranteed for certain SRT workloads. Various tradeoffs are possible with regard to the selection of $K(H)$. These tradeoffs are discussed in Section 4.5. After applying Theorem 3.2 to Subproblem 2, we have the following.

Corollary 4.1. *Let τ_s, M_s, $K(H)$, and G be as defined in Definition 4.2. The tardiness of any task $T_k \in \tau_s$ under a window-constrained scheduling policy is at most $\max(z, \rho) + e_k$, where*

$$z = \frac{E_L + \max\limits_{1 \leq \ell \leq |\tau_s|}(V(\ell))}{M_s - K(H) - G + \sum\limits_{h=1}^{K(H)+G} \widehat{u_h} - \max(K(H) + G - 1, 0) \max\limits_{1 \leq \ell \leq |\tau_s|}(u_\ell) - U_L}, \tag{4.8}$$

$$V(\ell) = e_\ell \cdot \left(\sum_{k=1}^{K(H)+G} (1 - \widehat{u_k}) - 1 \right) + 2 \cdot \sum_{k=1}^{K(H)+G} \widehat{u_k} \cdot \sigma_k$$
$$+ \sum_{T_k \in \tau_s \setminus T_\ell} \left(\left\lceil \frac{\rho + \mu}{p_k} \right\rceil + 1 \right) \cdot e_k + \min(M_s - K(H) - G, M_s - 1) \cdot \rho$$

provided (4.4) holds (with M replaced with M_s and τ replaced with τ_s) and (4.9) below holds.

$$M_s - K(H) - G + \sum_{h=1}^{K(H)+G} \widehat{u_h} - \max(K(H) + G - 1, 0) \max_{1 \leq \ell \leq |\tau_s|}(u_\ell) - U_L > 0 \tag{4.9}$$

Proof. We prove the corollary using results from Section 3.3; henceforth, when such results are applied, we assume that m is replaced with M_s and τ is replaced with τ_s. In the formulation of Subproblem 2, $K(H) + G$ supply functions $\beta_1^l(\Delta)$ may differ from Δ. Thus, $\overline{F} = K(H) + G$. By Definition 3.15 (see Section 3.3),

$$(\forall k : K(H) + G + 1 \leq k \leq M_s :: \sigma_k = 0 \wedge \widehat{u_k} = 1). \tag{4.10}$$

Thus,

$$\sum_{h=1}^{M_s} \widehat{u_h} = \sum_{h=1}^{K(H)+G} \widehat{u_h} + \sum_{h=K(H)+G+1}^{M_s} \widehat{u_h} = \sum_{h=1}^{K(H)+G} \widehat{u_h} + (M_s - K(H) - G), \tag{4.11}$$

$$\left(\sum_{k=1}^{M_s} (1 - \widehat{u_k}) - 1 \right) = \left(\sum_{k=1}^{K(H)+G} (1 - \widehat{u_k}) - 1 \right), \tag{4.12}$$

and

$$2 \cdot \sum_{k=1}^{M_s} \widehat{u_k} \cdot \sigma_k = 2 \cdot \sum_{k=1}^{K(H)+G} \widehat{u_k} \cdot \sigma_k. \tag{4.13}$$

Setting $\overline{F} = K(H) + G$ and substituting (4.12) and (4.13) into (3.34), we get $V(\ell)$ as defined in the statement of the corollary. Finally, substituting (4.11) into (3.33), we get z as defined in the statement of the corollary. $\square$

If GEDF is used for SRT tasks, then the tardiness bound in Corollary 4.1 can be further tightened by setting $V(\ell)$ in (4.8) to $e_\ell \cdot (\sum_{k=1}^{K(H)+G}(1 - \widehat{u_k}) - 1) + 2 \cdot \sum_{k=1}^{K(H)+G} \widehat{u_k} \cdot \sigma_k$, as shown in (3.37) in Section 3.4.

The following lemma shows that providing supply in MP form allows the widest range of SRT workloads to be supported.

Lemma 4.1. *If the supply to the tasks in τ_s is in MP form, then (4.9) always holds.*

Proof. If the supply to τ_s is in MP form, then $K(H) + G \leq 1$. We thus have

$$M_s - K(H) - G + \sum_{h=1}^{K(H)+G} \widehat{u_h} = M_s - 1 + \widehat{u_1}. \tag{4.14}$$

Setting $K(H) + G \leq 1$ and (4.14) into the left-hand side of (4.9) we have

88

$$M_s - K(H) - G + \sum_{h=1}^{K(H)+G} \widehat{u_h} - \max(K(H) + G - 1, 0) \max_{1 \le \ell \le |\tau_s|} (u_\ell) - U_L$$

$$= M_s - 1 + \widehat{u_1} - U_L. \tag{4.15}$$

We now consider two cases depending on the number of tasks in τ_s.

Case 1: $|\tau_s| \le M_s - 1$. In this case, by (4.6), $U_L = \sum_{i=1}^{|\tau_s|} u_i \le M_s - 1 < M_s - 1 + \widehat{u_1}$, where the latter inequality follows from Definition 1.2.

Case 2: $|\tau_s| > M_s - 1$. In this case, by (4.6), $U_L < U_{sum}(\tau_s) \overset{\{\text{by (4.4)}\}}{\le} \sum_{h=1}^{M_s} \widehat{u_h} = M_s - 1 + \widehat{u_1}$, where the latter equality follows from (4.10). The required result follows from (4.15) and the two cases above. $\square$

Corollary 4.2. *If at most one processor is partially available to τ_s, then Corollary 4.1 only requires that (4.4) holds. That is, bounded tardiness can be ensured with no utilization loss.*

Note that, if all $M_s \ge 2$ processors are fully available, then a HRT GEDF schedulability test (e.g., (Baruah, 2007; Bertogna et al., 2008; Baruah and Baker, 2008)) can be applied to τ before calculating tardiness bounds. If this test passes, then maximum tardiness is zero.

4.4.2 Computing Next-Level Supply

The remaining issue is to compute the supply of each child container in MP form, so that our analysis can be applied recursively in a container hierarchy. Note that we can do this regardless of whether the basic or extended approach described in Section 4.3 is used. Ensuring that child-container supplies are in MP form ensures that Property (P) holds for such containers.

If a server task $S_i(e_i, p_i)$ has bounded deadline tardiness, then the total guaranteed long-term supply to container C_i will be proportional to the long-term supply of $m(C_i)$ fully available processors, which can be described by a set of $m(C_i)$ supply functions equal to Δ, plus that of a partially available processor with bandwidth $u_i = e_i/p_i$. We are left with characterizing the processor time that is available to C_i when the server task S_i is scheduled.

The supply guaranteed to the server task S_i will depend on its parameters, e_i and p_i, and its tardiness. The latter depends on the scheduling algorithm used for SRT and server tasks, their parameters, and

(if extended approach is used) the amount of supply reclaimed on HRT-occupied processors. In the derivation of guaranteed supply, we use a definition from Section 3.3, which we repeat below.

Definition 3.5. Let $\mathsf{A}(T_{i,j}, t_1, t_2, \mathcal{Q})$ be the allocation of job $T_{i,j}$ during the interval $[t_1, t_2)$ in the schedule $\mathcal{Q}$. Let $\mathsf{A}(T_i, t_1, t_2, \mathcal{Q})$ be the allocation of task T_i during the interval $[t_1, t_2)$ in the schedule $\mathcal{Q}$.

Lemma 4.2. *Let Θ_i be the maximum deadline tardiness of the server task S_i's jobs in $\mathcal{Q}$. Then, the allocation $\mathsf{A}(S_i, 0, t, \mathcal{Q})$ satisfies the following.*

$$\mathsf{A}(S_i, 0, t, \mathcal{Q}) \leq u_i \cdot t + e_i \cdot (1 - u_i) \tag{4.16}$$

$$\mathsf{A}(S_i, 0, t, \mathcal{Q}) \geq u_i \cdot t - u_i \cdot \Theta_i - e_i \cdot (1 - u_i) \tag{4.17}$$

Proof. We first prove (4.16). Let $S_{i,k}$ be the latest job of S_i in schedule $\mathcal{Q}$ such that $r_{i,k} \leq t$. (Such a job exists because S_i is a periodic server task.) Then, by Definition 3.5, the allocation of S_i in $[0, t)$ is

$\mathsf{A}(S_i, 0, t, \mathcal{Q})$

   {because $S_{i,k}$'s successors do not execute before t in any schedule}

$\leq \mathsf{A}(S_{i,k}, 0, t, \mathcal{Q}) + \sum_{j < k} \mathsf{A}(S_{i,j}, 0, t, \mathcal{Q})$

   {because the worst-case execution time of S_i is e_i}

$\leq \mathsf{A}(S_{i,k}, 0, t, \mathcal{Q}) + \sum_{j < k} e_i$

   {because $S_{i,k}$ is not scheduled before $r_{i,k}$}

$\leq \min(e_i, t - r_{i,k}) + \sum_{j < k} e_i. \tag{4.18}$

The latter expression is maximized if the number of jobs of S_i released before $r_{i,k}$ is maximized, as shown in Figure 4.5(a). Therefore, (4.18) is maximized if $k = \left\lfloor \frac{t}{p_i} \right\rfloor + 1$ and $r_{i,k} = (k-1) \cdot p_i$. Setting these values into (4.18), we have

$$\mathsf{A}(S_i, 0, t, \mathcal{Q}) \leq \min\left(e_i, t - \left\lfloor \frac{t}{p_i} \right\rfloor \cdot p_i\right) + e_i \cdot \left\lfloor \frac{t}{p_i} \right\rfloor$$
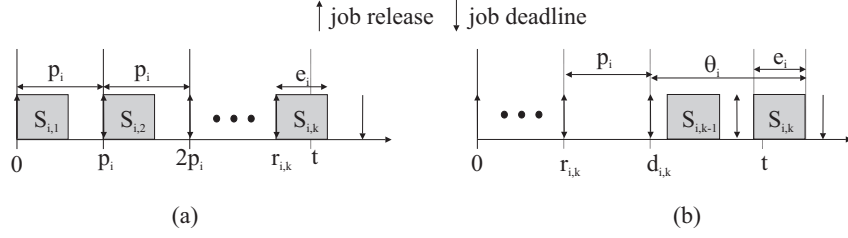
Figure 4.5: Server task's **(a)** maximum and **(b)** minimum allocation scenarios.

$$\{\text{setting } \tfrac{t}{p_i} = q\}$$

$$= \min\left(e_i, (q - \lfloor q \rfloor) \cdot p_i\right) + e_i \cdot \lfloor q \rfloor$$

$$= \min\left(e_i, (q - \lfloor q \rfloor) \cdot p_i\right) + e_i \cdot \lfloor q \rfloor + e \cdot q - e \cdot q$$

$$= \min\left(e_i \cdot (\lfloor q \rfloor - q + 1), (q - \lfloor q \rfloor) \cdot (p_i - e_i)\right) + e_i \cdot q$$

$$\{\text{setting } q - \lfloor q \rfloor = z\}$$

$$= \min\left(e_i \cdot (1 - z), z \cdot (p_i - e_i)\right) + e_i \cdot q$$

$$\left\{\begin{array}{l} \text{the } \min(\dots) \text{ summand is maximized when its two} \\ \text{arguments are equal, which is the case when } z = u_i \end{array}\right\}$$

$$\leq \min\left(e_i \cdot (1 - u_i), u_i \cdot (p_i - e_i)\right) + e_i \cdot q$$

$$\{\text{setting } q = \tfrac{t}{p_i}\}$$

$$= u_i \cdot t + e_i \cdot (1 - u_i).$$

We now prove (4.17). Let $S_{i,k}$ be the earliest job of S_i such that $d_{i,k} + \Theta_i \geq t$. For this job, since $d_{i,k} = r_{i,k} + p_i$, we have $r_{i,k} + p_i + \Theta_i \geq t$. Let

$$r_{i,k} = t - p_i - \Theta_i + \varepsilon, \tag{4.19}$$

where $\varepsilon \geq 0$. By the selection of $S_{i,k}$, for any job $S_{i,j}$ such that $j < k$, we have $d_{i,j} + \Theta_i < t$. By the statement of the lemma, each job of S_i completes within Θ_i time units after its deadline. Therefore, all jobs $S_{i,j}$ such that $j < k$ complete by time t, i.e.,

$$\mathsf{A}(S_{i,j}, 0, t, \mathcal{Q}) = e_i \text{ for each } j < k. \tag{4.20}$$

The allocation $\mathsf{A}(S_{i,k}, 0, t, \mathcal{Q})$ is minimized if $\mathsf{A}(S_{i,k}, t, d_{i,k} + \Theta_i, \mathcal{Q})$ is maximized. The latter is at most $\min(e_i, d_{i,k} + \Theta_i - t)$, as illustrated in Figure 4.5(b). Thus,

$$A(S_{i,k}, 0, t, \mathcal{Q}) = e_i - A(S_{i,k}, t, d_{i,k} + \Theta_i, \mathcal{Q})$$

$$\geq e_i - \min(e_i, d_{i,k} + \Theta_i - t)$$

$$= \max(0, e_i - (d_{i,k} + \Theta_i - t))$$

$$= \max(0, e_i - (r_{i,k} + p_i + \Theta_i - t))$$

$$\{\text{by } (4.19)\}$$

$$= \max(0, e_i - \varepsilon). \tag{4.21}$$

Since S_i's jobs are released periodically from time zero (since it is a server task), there are $\frac{r_{i,k}}{p_i}$ jobs released before job $S_{i,k}$. Thus,

$$A(S_i, 0, t, \mathcal{Q}) = A(S_{i,k}, 0, t, \mathcal{Q}) + \sum_{j<k}(A(S_{i,j}, 0, t, \mathcal{Q}))$$

$$\{\text{by } (4.20)\}$$

$$= A(S_{i,k}, 0, t, \mathcal{Q}) + \frac{r_{i,k}}{p_i} \cdot e_i$$

$$\{\text{by } (4.21)\}$$

$$\geq \max(0, e_i - \varepsilon) + \frac{r_{i,k}}{p_i} \cdot e_i$$

$$\{\text{by } (4.19)\}$$

$$= \max(0, e_i - \varepsilon) + \frac{t - p_i - \Theta_i + \varepsilon}{p_i} \cdot e_i$$

$$= \max(0, e_i - \varepsilon) + u_i \cdot t - u_i \cdot \Theta_i - e_i + \varepsilon \cdot u_i$$

$$= \max(u_i \cdot \varepsilon - e_i, \varepsilon \cdot (u_i - 1)) + u_i \cdot t - u_i \cdot \Theta_i$$

$$\left\{ \begin{array}{l} \text{the } \max(\dots) \text{ summand is minimized if its two} \\ \text{arguments are equal, which is the case when } \varepsilon = e_i \end{array} \right\}$$

$$\geq \max(u_i \cdot e_i - e_i, e_i \cdot (u_i - 1)) + u_i \cdot t - u_i \cdot \Theta_i$$

$$= u_i \cdot t - u_i \cdot \Theta_i - e_i \cdot (1 - u_i). \qquad \square$$

Example 4.8. Consider the schedule $\mathcal{Q}$ shown in Figure 4.3(b). In this schedule, jobs of the server task $S_1(1, 3)$ execute in the intervals $[0, 1)$, $[3, 4)$, and $[6, 7)$. By time 1, S_1 has received one allocation unit,

92
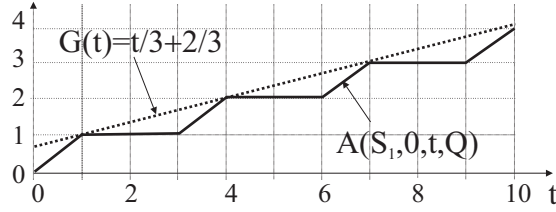
Figure 4.6: Server task allocation $\mathsf{A}(S_1, 0, t, \mathcal{Q})$ in Example 4.8 and its linear upper bound $G(t)$.

by time 4, its allocation is two units, and so on. The allocation $\mathsf{A}(S_1, 0, t, \mathcal{Q})$ is shown in Figure 4.6 as a function of t. The figure also shows the upper bound (4.16), which is $G(t) \triangleq u_i \cdot t + e_i(1 - u_i) = 1/3 \cdot t + 1(1 - 1/3) = 1/3 \cdot t + 2/3$. It is easy to see that $\mathsf{A}(S_1, 0, t, \mathcal{Q}) \leq G(t)$.

We now can find guarantees on the supplied processor time for server tasks for an arbitrary time interval.

Theorem 4.1. *Suppose that the scheduling algorithm used by the container H ensures a deadline tardiness bound of Θ_i for the server task $S_i(e_i, p_i)$. Then S_i is guaranteed at least $\eta_i^l(\Delta) = \max(0, u_i \cdot \Delta - 2 \cdot e_i \cdot (1 - u_i) - u_i \cdot \Theta_i)$ time units during an interval of length Δ.*

Proof. Our goal is to bound the allocation of S_i during an interval $[t_1, t_2)$ by a function of the length of the interval $\Delta = t_2 - t_1$.

$$\mathsf{A}(S_i, t_1, t_2, \mathcal{Q}) = \mathsf{A}(S_i, 0, t_2, \mathcal{Q}) - \mathsf{A}(S_i, 0, t_1, \mathcal{Q})$$

$$\{\text{by (4.16) and (4.17)}\}$$

$$\geq u_i \cdot t_2 - u_i \cdot \Theta_i - e_i \cdot (1 - u_i) - (u_i \cdot t_1 + e_i \cdot (1 - u_i))$$

$$= u_i \cdot (t_2 - t_1) - 2 \cdot e_i \cdot (1 - u_i) - u_i \cdot \Theta_i$$

$$= u_i \cdot \Delta - 2 \cdot e_i \cdot (1 - u_i) - u_i \cdot \Theta_i.$$

$\mathsf{A}(S_i, t_1, t_2, \mathcal{Q})$ cannot be less than zero, thus $\mathsf{A}(S_i, t_1, t_2, \mathcal{Q}) \geq \max(0, u_i \cdot \Delta - 2 \cdot e_i \cdot (1 - u_i) - u_i \cdot \Theta_i)$. □

Corollary 4.3. *The supply to container C_i, as defined above, is described by $M(C_i) = \lceil w(C_i) \rceil$ availability functions in MP form, where $m(C_i) = \lfloor w(C_i) \rfloor$ supply functions satisfy $\beta_j^l(\Delta) = \Delta$ and at most one supply function satisfies $\beta_1^l(\Delta) = \eta_i^l(\Delta)$ as given by Theorem 4.1. The total supplied bandwidth for C_i is $w(C_i)$.*

93

4.4.3 Computing Available Supply on HRT-Occupied Processors

In the previous section, we computed the supply available to a child container provided the tardiness bounds of tasks in τ_s are known. In order to calculate these tardiness bounds using Corollary 4.1, we need to determine the supply available to τ_s on $K(H)$ processors where HRT and SRT tasks are co-scheduled (if HRT capacity is reclaimed) in addition to the supply provided by the parent of H.

We first compute an upper bound on the allocation of an HRT task over the time interval $[t, t + \Delta)$.

Lemma 4.3. *If jobs of T_i finish by their deadlines in the schedule $\mathcal{Q}$, then $\mathsf{A}(T_i, t, t + \Delta, \mathcal{Q}) \leq u_i \cdot \Delta + 2 \cdot e_i \cdot (1 - u_i)$, for any t and $\Delta \geq 0$.*

Proof. Let $T_{i,k}$ be job of T_i with smallest index k that executes within $[t_1, t_2)$. If no such job exists, then T_i's allocation within $[t, t + \Delta)$ is zero and the required result holds trivially. Let $f_{i,k}$ be $T_{i,k}$'s completion time. The allocation of $T_{i,k}$ is thus

$$\mathsf{A}(T_{i,k}, t, t + \Delta, \mathcal{Q}) \leq \min(e_i, \Delta, \varepsilon), \tag{4.22}$$

where $\varepsilon = f_{i,k} - t$, as illustrated in Figure 4.7. We consider two cases based upon the relationship between ε and Δ.

Case 1: $\varepsilon > \Delta$. In this case, $T_{i,k}$ commences execution at or before $t + \Delta$ and finishes after $t + \Delta$. By the selection of k, $T_{i,k}$ is the only job of T_i that executes within $[t, t + \Delta)$. Therefore, T_i's allocation in this interval cannot be greater than $\min(e_i, \Delta)$. By (4.22) and the condition of Case 1, we have

$$
\begin{aligned}
\mathsf{A}(T_{i,k}, t, t + \Delta, \mathcal{Q}) \leq\ & \min(e_i, \Delta) \\
=\ & u_i \cdot \min(e_i, \Delta) + (1 - u_i) \cdot \min(e_i, \Delta) \\
\leq\ & u_i \cdot \Delta + (1 - u_i) \cdot e_i \\
\leq\ & u_i \cdot \Delta + 2 \cdot (1 - u_i) \cdot e_i.
\end{aligned}
$$

Case 2: $\varepsilon \leq \Delta$. Because, by the condition of the lemma, $T_{i,k}$ finishes by its deadline, $f_{i,k} \leq d_{i,k} = r_{i,k} + p_k \leq r_{i,k+1}$. The allocation of $T_{i,k}$'s successor jobs in the interval $[t, t + \Delta)$ is maximized if all of these jobs are released as soon as possible after $f_{i,k}$, as shown in Figure 4.7. Therefore,

$$\sum_{j > k} \mathsf{A}(T_{i,j}, t, t + \Delta, \mathcal{Q}) \leq \max\left(0, \left\lfloor \frac{t + \Delta - f_{i,k}}{p_i} \right\rfloor \cdot e_i + \min(e_i, (t + \Delta - f_{i,k}) \bmod p_i)\right)$$

$$\{\text{setting } f_{i,k} - t = \varepsilon\}$$

$$= \max\left(0, \left\lfloor \frac{\Delta - \varepsilon}{p_i} \right\rfloor \cdot e_i + \min(e_i, (\Delta - \varepsilon) \bmod p_i)\right). \qquad (4.23)$$

By Definition 3.5 and the selection of k,

$$\mathsf{A}(T_i, t, t + \Delta, \mathcal{Q}) = \mathsf{A}(T_{i,k}, t, t + \Delta, \mathcal{Q}) + \sum_{j > k} \mathsf{A}(T_{i,j}, t, t + \Delta, \mathcal{Q})$$

$$\{\text{by (4.22) and (4.23)}\}$$

$$\leq \min(e_i, \Delta, \varepsilon) + \max\left(0, \left\lfloor \frac{\Delta - \varepsilon}{p_i} \right\rfloor \cdot e_i + \min(e_i, (\Delta - \varepsilon) \bmod p_i)\right)$$

$$\{\text{by the condition of Case 2}\}$$

$$= \min(e_i, \varepsilon) + \left\lfloor \frac{\Delta - \varepsilon}{p_i} \right\rfloor \cdot e_i + \min(e_i, (\Delta - \varepsilon) \bmod p_i)$$

$$\left\{\text{setting } \frac{\Delta - \varepsilon}{p_i} = q\right\}$$

$$= \min(e_i, \varepsilon) + \lfloor q \rfloor \cdot e_i + \min(e_i, q \cdot p_i - \lfloor q \rfloor \cdot p_i)$$

$$= \min(e_i, \varepsilon) + q \cdot e_i - q \cdot e_i + \lfloor q \rfloor \cdot e_i + \min(e_i, q \cdot p_i - \lfloor q \rfloor \cdot p_i)$$

$$= \min(e_i, \varepsilon) + q \cdot e_i + \min(e_i \cdot (\lfloor q \rfloor - q + 1), (q - \lfloor q \rfloor) \cdot (p_i - e_i))$$

$$\{\text{setting } q - \lfloor q \rfloor = z\}$$

$$= \min(e_i, \varepsilon) + q \cdot e_i + \min(e_i \cdot (1 - z), z \cdot (p_i - e_i))$$

$$\left\{\begin{array}{l} \min(e_i \cdot (1 - z), z \cdot (p_i - e_i)) \text{ is maximized if both its} \\ \text{arguments are equal, which is the case when } z = u_i \end{array}\right\}$$

$$\leq \min(e_i, \varepsilon) + q \cdot e_i + \min(e_i \cdot (1 - u_i), u_i \cdot (p_i - e_i))$$

$$\left\{\text{setting } q = \frac{\Delta - \varepsilon}{p_i}\right\}$$

$$= \min(e_i, \varepsilon) + \frac{\Delta - \varepsilon}{p_i} \cdot e_i + e_i \cdot (1 - u_i)$$

$$= \min(e_i, \varepsilon) + (\Delta - \varepsilon) \cdot u_i + e_i \cdot (1 - u_i)$$

$$\{\text{maximized if } \varepsilon = e_i\}$$

$$\leq e_i + (\Delta - e_i) \cdot u_i + e_i \cdot (1 - u_i)$$

$$= u_i \cdot \Delta + 2 \cdot e_i \cdot (1 - u_i). \qquad \qquad \square$$
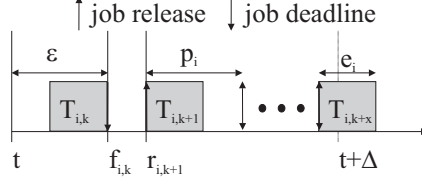
Figure 4.7: Maximum allocation scenario for a HRT task T_i.

Lemma 4.4. *Let τ_h be the set of HRT tasks assigned to a fully available processor h such that $U_{sum}(\tau_h) < 1$. For any time interval of length Δ, at least $\beta_h^l(\Delta) = \max(0, \widehat{u_h} \cdot (\Delta - \sigma_h))$ time units are available, where $\widehat{u_h} = 1 - U_{sum}(\tau_h)$ and $\sigma_h = \frac{2\sum_{T_i \in \tau_h} e_i \cdot (1 - u_i)}{1 - U_{sum}(\tau_h)}$.*

Proof. Consider an interval $[t, t + \Delta)$. By Definition 3.5, the time available after scheduling τ_h within this interval is

$$\max\left(0, \Delta - \sum_{T_i \in \tau_h} \mathsf{A}(T_i, t, t + \Delta)\right)$$

$$\{\text{setting } t_1 = t \text{ and } t_2 = t + \Delta \text{ into Lemma 4.3}\}$$

$$\geq \max\left(0, \Delta - \sum_{T_i \in \tau_h} (u_i \cdot \Delta + 2 \cdot e_i \cdot (1 - u_i))\right)$$

$$\{\text{by Definition 1.1}\}$$

$$\geq \max\left(0, \Delta \cdot (1 - U_{sum}(\tau_h)) - 2 \cdot \sum_{T_i \in \tau_h} e_i \cdot (1 - u_i)\right)$$

$$\{\text{by the definition of } \widehat{u_h} \text{ and } \sigma_h \text{ in the statement of the lemma}\}$$

$$= \max(0, \widehat{u_h} \cdot (\Delta - \sigma_h)). \qquad \square$$

Definition 4.3. Let $M(H)$ be the total number of processors that provide supply to H. Let $Y = M(H) - \sum_{C_j \in succ(H)} m(C_j) - m(C_{hrt})$ be the number of processors that are not reserved for HRT tasks and child containers of H.

The following theorem summarizes the analysis discussed in the previous sections. In the statement of the theorem, G, $K(H)$, and τ_s are as defined earlier in Definitions 4.1 and 4.2, and $M_s = Y + K(H)$.

Theorem 4.2. *If the host container H's supply is in MP form, then hard real-time schedulability for HRT tasks and bounded deadline tardiness for SRT and server tasks encapsulated in H are guaranteed if (4.4) holds (with M is replaced with M_s and τ is replaced with τ_s) and (4.9) holds. If deadline tardiness is bounded for a server task, then the supply to the corresponding child container is in MP form and the*
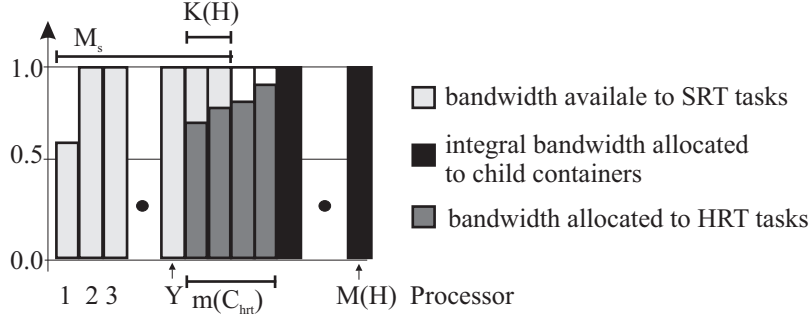
Figure 4.8: Illustration of Theorem 4.2.

supplied bandwidth matches that specified for the child container.

Proof. We illustrate the proof using Figure 4.8. In this figure, the supply available to H is represented as $M(H)$ bins for which the height of the bin represents the available utilization on the respective processor. We first dedicate an integral number of processors to supply the integral part of the child containers' bandwidths (these processors are shaded black). We then partition the tasks in $\mathsf{HRT}(H)$ among $m(C_{hrt})$ processors and find the number Y as defined in Definition 4.3. For each processor h such that $h \in [Y+1, Y+m(C_{hrt})]$, we find the unused bandwidth $\widehat{u_h} = 1 - U_{sum}(\tau_h)$ using Lemma 4.4, as shown in Figure 4.8. After determining $K(H)$, we find $M_s = Y + K(H)$ and the bandwidth available to SRT and server tasks (this bandwidth is shaded light gray in Figure 4.8). In order to apply Corollary 4.1, we need to re-number the processors with indices 1 to M_s so that partially available processors are listed first. Finally, we apply Corollary 4.1 to calculate tardiness bounds for the tasks in τ_s and use Corollary 4.3 to find the supply functions for child containers. Each child container $C_j \in succ(H)$ is thus guaranteed supply from an integral number of fully available processors plus the time allocated on an additional processor whenever the respective server task S_j is scheduled. By Corollary 4.3, this allocation is proportional to S_j's utilization, which is the fractional part of C_j's bandwidth. Therefore, the supplied bandwidth to each child container C_j is proportional to its required bandwidth and is in MP form. $\qquad\square$

Applying the above theorem recursively, we can analyze the properties of a container hierarchy. Note that the tardiness of SRT tasks may be higher as compared to a corresponding non-hierarchical approach, where all tasks are scheduled at the same level because the degree of parallelism of the available supply is lower under our approach. This is the price for having temporal isolation among containers. In Section 4.6, we discuss in greater detail the conditions under which temporal isolation is guaranteed.

97
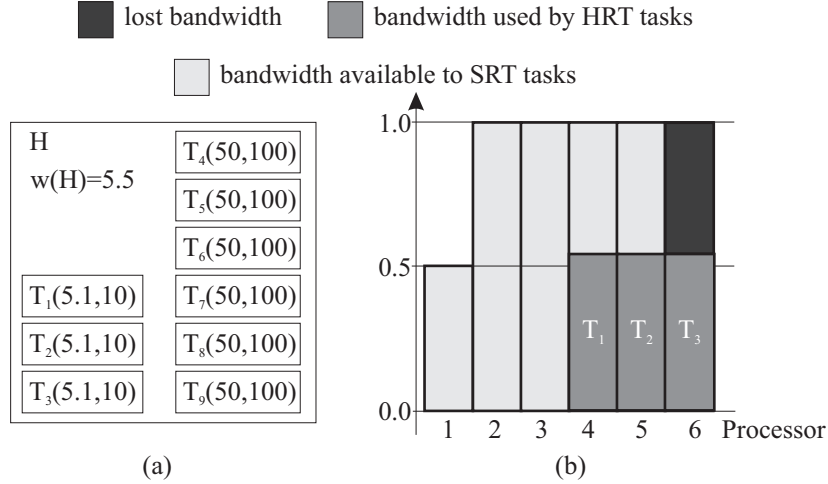
Figure 4.9: Bandwidth allocation and utilization loss in Example 4.9.

4.5 Tradeoffs for HRT Tasks

If there are no HRT tasks in the system, then no utilization loss is incurred. If the system has HRT tasks, then tradeoffs between the schedulability and tardiness of SRT tasks and utilization loss are possible, as illustrated by the example below.

Example 4.9. Consider a container H encapsulating three HRT tasks T_1, T_2, and T_3 with utilization 0.51 and six SRT tasks $\mathsf{SRT}(H) = \{T_4, \ldots, T_9\}$ with utilization 0.5 as shown in Figure 4.9(a). H's bandwidth of $w(H) = 5.5$ is supplied by a partially available processor 1 with $\widehat{u_1} = 0.5$ and five fully available processors, as shown in Figure 4.9(b). In this figure, the processors are represented as six bins. By (4.2), the HRT tasks require three dedicated processors since no two of these tasks can be assigned to one processor without violating HRT constraints. These tasks are therefore assigned to processors 4–6. The bandwidth consumed by the HRT tasks is shaded. After the HRT tasks are allocated, the total bandwidth provided by processors 1–3, which is 2.5, is insufficient to handle all SRT tasks, whose total utilization is $U_{sum}(\mathsf{SRT}(H)) = 3$. We reclaim the unused bandwidth on processors 4 and 5 by setting $K(H) = 2$ (see Definition 4.1). The supply available to the SRT tasks is now given by $M_s = 5$ processors with utilizations $\widehat{u_1} = 0.49$, $\widehat{u_2} = 0.49$, $\widehat{u_3} = 0.5$, $\widehat{u_4} = 1.0$, and $\widehat{u_5} = 1.0$, respectively. (Note that processors are ordered by increasing utilizations. The first two utilization values were obtained using Lemma 4.4.) The total supplied bandwidth is thus $\sum_{k=1}^{M_s} \widehat{u_k} = 3.48$, which exceeds the total utilization of the SRT tasks, and hence, (4.4) holds. Because the supply to the SRT tasks is not in MP form (i.e., more than one processor is partially available), by Corollary 4.1, we have to test whether (4.9) holds in order to check the schedulability of $T_4, \ldots, T_9$. Setting the supply and task parameters into (4.9), we

have

$$M_s - K(H) - G + \sum_{h=1}^{K(H)+G} \widehat{u_h} - \max(K(H) + G - 1, 0) \max_{1 \leq \ell \leq |\tau_s|} (u_\ell) - U_L$$

$$\left\{ \begin{array}{l} \text{because } M_s = 5, \, G = 1, \, K(H) = 2, \, \max(u_\ell) = 0.5, \\[4pt] \text{and } U_L = (M_s - 1) \cdot 0.5 = 2 \end{array} \right\}$$

$$= 5 - 2 - 1 + (0.49 + 0.49 + 0.5) - \max(3 - 1, 0) \cdot 0.5 - 2$$

$$= 0.48$$

$$> 0.$$

Thus, bounded tardiness for the SRT tasks is guaranteed if $K(H) = 2$. Also, the utilization loss, which is the bandwidth that is unused by HRT tasks and that is unavailable to SRT tasks, is 0.49 in this case (this unused utilization is shaded black in Figure 4.9(b)). If we try to reduce the utilization loss even further by setting $K(H)$ to 3, then, even though the total utilization available to the SRT tasks becomes 3.97, (4.9) no longer holds.

The example above shows that the co-scheduling of HRT and SRT tasks may be necessary in order to accommodate a workload using the supplied bandwidth. However, SRT schedulability can be compromised for large $K(H)$ due to (4.9). To find the maximum $K(H)$ so that the tasks in τ_s remain schedulable, we can apply Theorem 4.2 for each $K(H)$ from $m(C_{hrt})$ to zero.

From (4.9) and (4.6), we conclude that (4.9) is more likely to hold if $K(H)$ or $\max_{1 \leq \ell \leq |\tau_s|}(u_\ell)$ is small. Therefore, reclaiming processor time can be successful if the maximum per-task utilization of SRT and server tasks is small.

4.6   Misbehaving Tasks

We call a task T_i *misbehaving* if its worst-case execution time may exceed e_i. In this section, we describe the impact of misbehaving tasks on a system and show how to alleviate any adverse effects. Consider the container configuration shown in Figure 4.10. In this figure, T_1 is a misbehaving task and is denoted by a star-shaped outline. In the configuration shown in Figure 4.10, the processor supplies of C_1 and C_3 depend solely on the supply of H and the parameters of the server tasks S_1 and S_3, which cannot be misbehaving since a server task is not scheduled when its budget is depleted. By Corollary 4.3, the parameters of S_3 and its deadline tardiness define the guaranteed supply of C_3, and hence, the tardiness
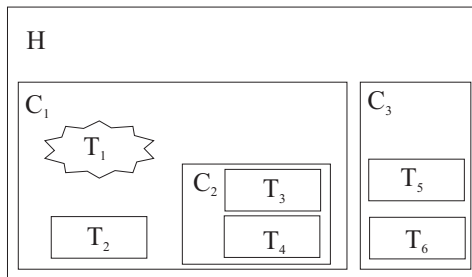
Figure 4.10: Container isolation.

of T_5 and T_6. Thus, the misbehaving task T_1 does not affect the timeliness of tasks belonging to C_3. That is, the tasks in container C_3 are *temporally isolated* from the misbehaving task. More generally, any two tasks $T_i \in succ(C_k)$ and $T_j \in succ(C_l)$ are temporally isolated iff C_k is not a member of the hierarchy rooted at C_l and C_l is not a member of the hierarchy rooted at C_k.

On the other hand, a misbehaving task T_i *can* affect the timeliness of tasks encapsulated in that part of container hierarchy that is rooted at T_i's parent. In our example, due to the misbehaving task T_1, task T_2's tardiness may exceed its computed bound. As a consequence, the tardiness of the server task S_2 of container C_2 may exceed its computed bound thereby invalidating the bounds on processor allocation for container C_2. This, in turn, may affect the timeliness of the encapsulated tasks T_3 and T_4. To prevent such problems, any potentially misbehaving task should be isolated in a container for which a budget can be enforced.

4.7 Experiments

We now present the results of experiments conducted to compare our container-aware scheduling scheme with conventional scheduling techniques. In these experiments, performance was compared using randomly-generated task sets, which have both HRT and SRT tasks.

Task generation procedure. In order to gain intuition about the properties of a large multiprocessor platform running multiple isolated components, we evaluated a three-level container hierarchy consisting of a root container C_0, four second-level containers, and then the contained tasks, as shown in Figure 4.11. The i-th second-level container is denoted $C_{sys}^{[i]}$ and its contained HRT and SRT tasks as $\tau_{hrt}^{[i]}$ and $\tau_{srt}^{[i]}$, respectively. Randomly-generated tasks were added to these sets while $U(\tau_{hrt}^{[i]})$ is at most $U_{hrt} \leq 1$ and $U(\tau_{srt}^{[i]}) \leq 3.5$. Task utilizations were taken randomly from $[0, 0.15)$ for HRT tasks and from $[u_{min}, u_{max})$ for SRT tasks. We examined three HRT total utilization caps U_{hrt} and four SRT utilization ranges, as described later. Integral task periods were taken randomly from $[100, 1000]$ for HRT tasks and from
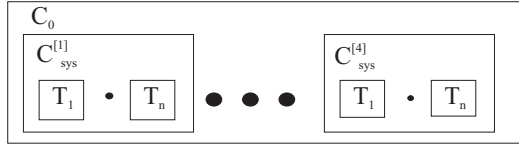
Figure 4.11: Experimental setup.

[10000, 50000] for SRT tasks. Integral execution times were computed using periods and utilizations.

We compared our container-aware scheduling scheme (Container-Aware Scheduling (CA)) with PEDF and a hybrid EDF-based scheme (Hard-Soft Scheduling (HS)), both of which are oblivious to containers. The HS scheme, which is described later in this section, is a naïve combination of PEDF and GEDF. PEDF was selected because it exhibits good timeliness, and HS was selected because it can satisfy the requirements of HRT and SRT tasks using relatively few processors. However, HS and PEDF do not provide any isolation among containers. In our experiments, we compared the tested schemes based on *the required number of processors* (RNP) and *deadline tardiness bound* (TB). We did not consider any system overheads or other container hierarchies. Such things are very application- and implementation-specific, respectively, and our intent here is only to provide a basic sense of how our scheme compares to the other implementation alternatives.

Defining RNP. Under PEDF, RNP is defined as the minimum number of processors required to partition all real-time tasks using the first-fit heuristic. Under PEDF, all tasks have zero tardiness.

Under HS, HRT and SRT tasks run on disjoint processor sets, with all HRT tasks scheduled together using PEDF with the first-fit heuristic, and all SRT tasks scheduled together using GEDF. RNP for the SRT tasks is thus

$$M_{soft} = \left\lceil \sum_{i=1}^{4} U_{sum}(\tau_{srt}^{[i]}) \right\rceil .$$

Letting M_{hard} denote the HRT RNP, overall RNP under HS is simply $M_{hard} + M_{soft}$.

Under CA, we set container $C_{sys}^{[i]}$'s bandwidth to $w(C_{sys}^{[i]}) = W_I + W_f$ where W_I is the number of required fully available processors, and W_f is the minimum utilization due to (at most one) partially available processor. As explained next, W_I and W_f were determined based upon whether it is possible to reclaim bandwidth not used by HRT tasks (we illustrate this explanation with an example below). Because $U(\tau_{hrt}^{[i]}) \leq U_{hrt} \leq 1$, the HRT tasks of each second-level container require at most one processor. We checked whether any bandwidth on this processor can be reclaimed for SRT tasks as follows. We set $K_r(H) = 1$ (reclaiming is possible) if $\tau_{srt}^{[i]}$ is schedulable on $\left\lceil U_{sum}(\tau_{hrt}^{[i]} \cup \tau_{srt}^{[i]}) \right\rceil$ processors such that one processor has an available utilization of $1 - U_{sum}(\tau_{hrt}^{[i]})$ and one processor has an available utilization

of $\mathsf{frac}(U_{sum}(\tau_{hrt}^{[i]} \cup \tau_{srt}^{[i]}))$, where $\mathsf{frac}(x)$ is the fractional part of x. Otherwise, we set $K_r(H) = 0$ (i.e., reclaiming is not possible). After the degree of reclamation was determined, we set

$$W_I = \begin{cases} \lfloor U_{sum}(\tau_{srt}^{[i]} \cup \tau_{hrt}^{[i]}) \rfloor & \text{if } K_r(H) = 1 \\ \lfloor U_{sum}(\tau_{srt}^{[i]}) \rfloor + 1 & \text{otherwise.} \end{cases}$$

The fractional part of the bandwidth W_f was set to

$$W_f = \begin{cases} \mathsf{frac}(U_{sum}(\tau_{srt}^{[i]} \cup \tau_{hrt}^{[i]})) & \text{if } K_r(H) = 1 \\ \mathsf{frac}(U_{sum}(\tau_{srt}^{[i]})) & \text{otherwise.} \end{cases}$$

Example 4.10. Consider container $C_{sys}^{[1]}$ with HRT task $T_1(200, 300)$ and SRT tasks $T_2(100, 400), \ldots,$ $T_4(100, 400)$, and $T_5(500, 800)$ as shown in Figure 4.12(a). (Note that these task parameters are not allowed by out task generation method; however, allowing them simplifies the example.) For this task set, $U_{sum}(\tau_{hrt}^{[1]}) = 2/3$, $U_{sum}(\tau_{srt}^{[1]}) = 11/8$, and $U_{sum}(\tau_{srt}^{[1]} \cup \tau_{hrt}^{[1]}) = 49/24$. We first check the schedulability of $T_2, \ldots, T_5$ on $\left\lceil U_{sum}(\tau_{srt}^{[1]} \cup \tau_{hrt}^{[1]}) \right\rceil = 3$ processors such that one processor is fully available and two processors have available utilizations of $1/24$ and $1/3$ (see processors 1 and 3 in Figure 4.12(b)). It can be shown that (4.9) does does not hold for this task system, and hence, we have to set $K(H) = 0$. With this setting of $K(H)$, we cannot co-schedule the HRT and the SRT tasks on processor 3. It can be verified that the SRT tasks are schedulable on $\lceil U_{sum}(\tau_{srt}^{[1]}) \rceil = 2$ processors such that one processor is fully available and one processor has an available utilization of $\mathsf{frac}(U_{sum}(\tau_{srt}^{[1]})) = 3/8$ (see processors 1 and 2 in Figure 4.12(b)). Therefore, we set $W_I = 2$, since the HRT and the SRT tasks together require two fully available processors, and $W_f = 3/8$, because the SRT tasks additionally need a bandwidth of $\mathsf{frac}(U_{sum}(\tau_{srt}^{[1]})) = 3/8$.

The execution time e_i and the period p_i of the server task $S_{srt}^{[i]}$ should be set such that $e_i/p_i = W_f$. Once W_f has been determined and a value for e_i is selected, p_i is implicitly determined. However, a tradeoff exists in selecting e_i. On one hand, a smaller value of e_i effectively reduces the server task's maximum tardiness, and correspondingly, the supply blackout time, as (4.8) and Theorem 4.1 suggest. On the other hand, small server task execution times could lead to frequent context switches in a real implementation. As a compromise, we set the execution time of each server task to be 100, which is close to the average execution time of SRT tasks in H. Server tasks' periods were set to $\left\lfloor \frac{100}{W_f} \right\rfloor$ if $W_f \neq 0$, so that the utilization of the server task is slightly higher than the fractional part of the required container
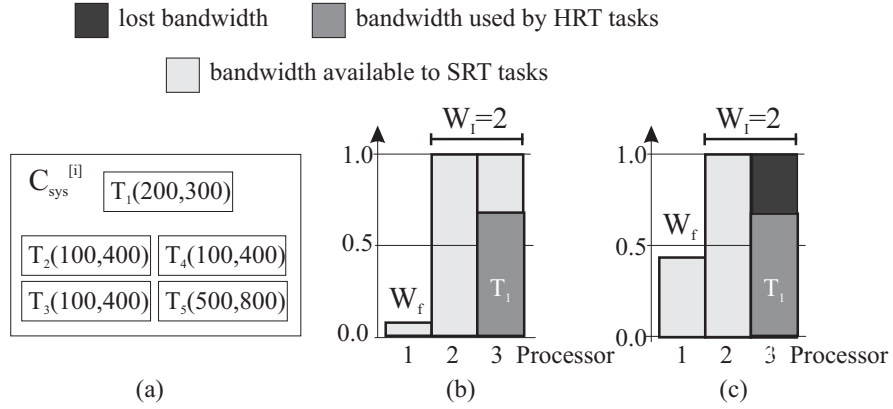
Figure 4.12: Determining the required container bandwidth in Example 4.10.

bandwidth. The required container bandwidth $C_{sys}^{[i]}$ was then inflated accordingly by

$$
u_S = \begin{cases} 0 & \text{if } W_f = 0 \\ \frac{100}{\lceil 100/W_f \rceil} - W_f & \text{otherwise,} \end{cases}
$$

where u_S is the utilization loss associated with the choice of server task parameters.

As an example consider container $C_{sys}^{[i]}$ from Example 4.10. Because $W_f = 3/8$, we set $u_S = \frac{100}{\lceil 100/(3/8) \rceil} - 3/8 = 0.001$.

Overall, RNP for CA is simply the bandwidth of the root container C_0, $w(C_0) = \lceil \sum_{i=1}^{4} w(C_{sys}^{[i]}) \rceil$.

RNP results. Insets (a), (c), (e), and (g) of Figure 4.13 show RNP results for PEDF, HS, and CA, for the SRT utilization ranges $[0.01, 0.1)$ (light), $[0.1, 0.5)$ (medium), and $[0.5, 1)$ (heavy), respectively. We also examined the SRT utilization range $[0.5, 0.7)$ (extreme) as well, as it is an extreme case where PEDF shows poor performance. The x axis in each inset corresponds to the HRT utilization cap, U_{hrt}.

For each utilization range, 100 task sets were generated and their RNP averaged. The figure also shows the average total system utilization, so that we can estimate the utilization loss associated with each scheme.

For the light and medium SRT per-task utilization ranges (insets (a) and (c)), all three schemes show similar performance. This is because CA is able to minimize the bandwidth of individual second-level containers by co-scheduling HRT and SRT tasks together. As SRT per-task utilization increases, RNP for PEDF also increases because more processors are needed to bin-pack the SRT tasks. The extreme case (inset (g)) is the utilization range $[0.5, 0.7)$, where each SRT task requires a separate processor.

When SRT per-task utilizations are large (inset (e)), the difference between HS and CA is maximal,

due to the utilization loss associated with HRT tasks in the containers. Under CA, the four HRT task sets require four processors, while under HS, all HRT tasks may be packed onto a smaller number of processors.

Tardiness.   Insets (b), (d), (f), and (h) of Figure 4.13 show the average of the per-task-set tardiness bounds under HS and CA for the task set categories discussed above (under PEDF, tardiness is zero). For these two schemes, these tardiness bounds are comparable in most cases, with the tardiness under CA being slightly higher due to uneven supply by the server tasks. Under CA, the maximum tardiness bound is significantly higher when the maximum total utilization of HRT tasks is high (see the HRT utilization cap of 0.9 in insets (b) and (d) of Figure 4.13). This is because CA attempts to reclaim scarce processor supply available after scheduling HRT tasks within the container and use that supply to schedule SRT tasks. However, even though the maximum task tardiness in these cases is higher, the number of processors required by CA is lower (see insets (a) and (c) of Figure 4.13).

Overall, these experiments show that in some cases there is a price to be paid for temporal isolation among containers, in the form of more required processors (if HRT tasks are present) or higher tardiness. However, in our proposed scheme, this price is reasonable, when considering the performance of schemes that ensure no isolation. As a final comment, we remind the reader that if no HRT tasks are present, then our scheme incurs *no* utilization loss.

4.8   Summary

In this chapter, we have presented a multiprocessor bandwidth-reservation scheme for hierarchically organized real-time containers. Under this scheme each real-time container can reserve any fraction of processor time (even the capacity of several processors) to schedule its children. The presented scheme provides temporal isolation among containers so that each container can be analyzed separately.

Our scheme is novel in that soft real-time components incur no utilization loss. This stands in sharp contrast to hierarchical schemes for hard (only) real-time systems, where the loss per level can be so significant, arbitrarily deep hierarchies simply become untenable.

The most important for future work is to enable dynamic container creation and the joining/leaving of tasks. Also of importance is the inclusion of support for synchronization. It would also be interesting to investigate other global scheduling algorithms such as Pfair algorithms to see whether a more accurate analysis can be established for them.
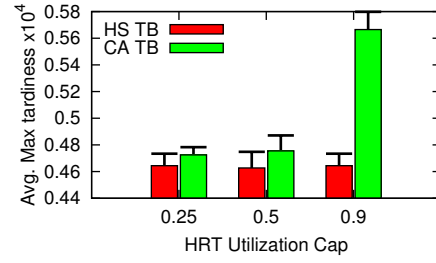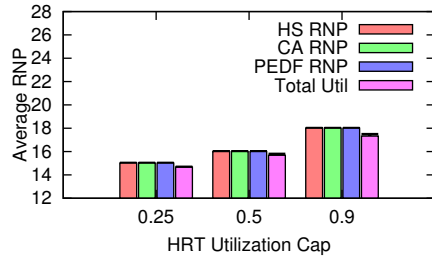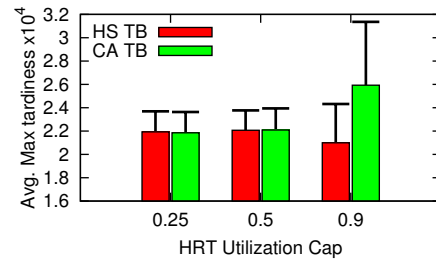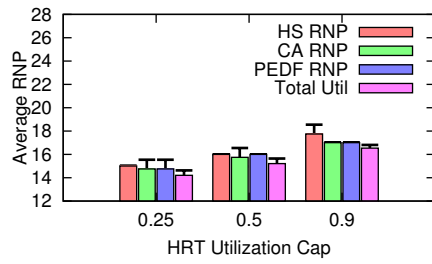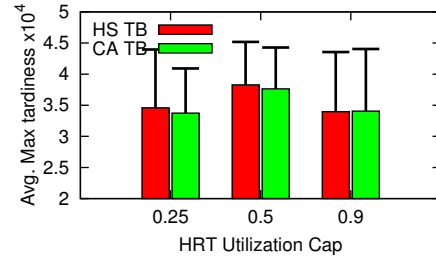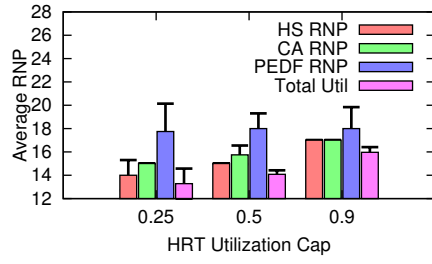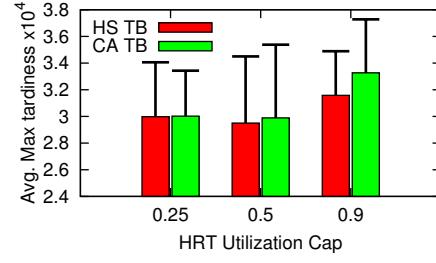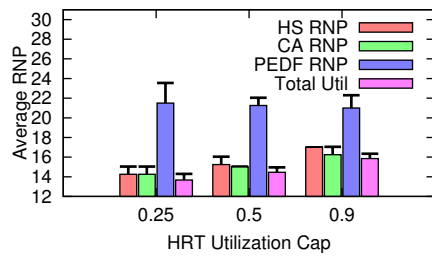
Figure 4.13: **(a,c,e,g)** Required number of processors and **(b,d,f,h)** maximum tardiness bounds for randomly generated task sets (with 95% confidence intervals) for **(a)**–**(b)** light, **(c)**–**(d)** medium, **(e)**–**(f)** heavy, and **(g)**–**(h)** extreme SRT utilization distributions.

Chapter 5

Multiprocessor Extensions to Real-Time Calculus

As mentioned in Section 1.7, the real-time calculus framework has been successfully used for the analysis of distributed and embedded systems. Unfortunately, it is only applicable to systems where partitioned scheduling algorithms are used. In this chapter, we present an extension of real-time calculus that enables the analysis of streaming task sets scheduled on a symmetric multiprocessor where the constituent processors are managed by a global scheduling algorithm.

The application of our results involves several theorems stated later and is illustrated in Figure 5.1. The core of our framework is a pseudo-polynomial-time procedure that, given a collection of arrival curves for input streams $\alpha_i^u(\Delta)$ and $\alpha_i^l(\Delta)$ (describing minimum and maximum number of arriving events over an interval of length Δ), their execution requirements, and the available resource supply $\mathcal{B}(\Delta)$, checks that event delays on such a multiprocessor reside within specified bounds. The set of delay bounds $\{\Theta_1, \ldots, \Theta_n\}$, where n is the number of streams, can be:

- specified (e.g., as relative deadlines) for individual tasks;

- calculated using Theorem 5.7 from the input if task deadlines are not specified or not feasible;

- determined by other means.

We should note that Theorem 5.7, which can be used to derive event-delay bounds from inputs, is only applicable in settings in which fixed-job-priority schedulers such as EDF or FIFO are used. When other schedulers are used, maximum delay bounds should be specified or found using alternative analysis techniques.
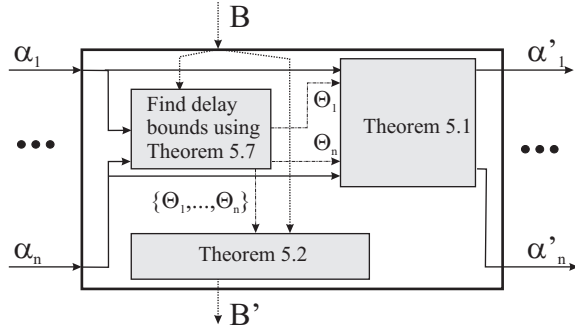
Figure 5.1: A multiprocessor PE analyzed using multiprocessor real-time calculus.

In terms of computational complexity, calculating delay bounds using Theorem 5.7 (where applicable) and comparing those to specified deadlines is less costly than checking whether specified deadline constraints are met using the pseudo-polynomial shedulability test.

As Theorem 5.7 gives somewhat conservative estimations of delay bounds, they might be further tightened by iteratively decreasing them and applying the schedulability test. However, the details of this tightening procedure are specific to a task set and running a schedulability test multiple times might be time-consuming. In our case study (described later), tightening the bounds obtained using Theorem 5.7 did not give much improvement.

Once the event delays are identified, we can compute arrival curves for the processed streams $\alpha^{u\prime}(\Delta)$ and $\alpha^{l\prime}(\Delta)$ using Theorem 5.1. This is done by algebraic manipulations involving the specified input curves. As a result, because the maximum event-delay is bounded, the long-term departure rate of events is the same as the long-term arrival rate, i.e., the long-term growth rate of $\alpha_i^{u\prime}(\Delta)$ is the same as that of $\alpha_i^u(\Delta)$ for each task T_i.

Also, we can compute the remaining-total-service curve $\mathcal{B}'(\Delta)$ using Theorem 5.2. In this case, we subtract the total execution demand of tasks within an interval of length Δ from the total available supply $\mathcal{B}(\Delta)$. The obtained output curves — as in the uniprocessor case — can in turn be used as input for other resources, thereby resulting in a compositional framework (as shown in Figures 1.6(a) and 1.9).

The rest of the chapter is organized as follows. Section 5.1 presents our task model. In Sections 5.2 and 5.3, the timing characteristics of processed streams and the remaining supply are computed. In Section 5.4, we present a basic response-time bound test. In Section 5.5, its time complexity is discussed. In Section 5.6, we improve the basic test for the case when an EDF-like scheduler is used. In Section 5.7, closed-form expressions for response-time bounds are derived. Section 5.8 presents a case study for our analysis. Section 5.9 summarizes our contributions and discusses some directions for future work.

107

Table 5.1: Model notation.

Input parameters	
$\alpha_i^u(\Delta)\ (\alpha_i^l(\Delta))$	Max. (min.) number of job arrivals of T_i over Δ
$\gamma_i^u(k)\ (\gamma_i^l(k))$	Max. (min.) execution demand of any k consecutive jobs of T_i
$\mathcal{B}(\Delta)$	Min. guaranteed cumulative processor supply over Δ
Params. below can be found using the RTC Toolbox	
$\widehat{U}$	Long-term avilable processor utilization
σ_{tot}	Maximum blackout time
F	Number of processors that are always available
$\mathcal{A}_i^{-1}(k)$	Pseudo-inverse of α_i^u
K_i	Min. integer s.t. $\mathcal{A}^{-1}(K_i) \geq \gamma_i^u(K_i)$
$\overline{e_i}$	T_i's average worst-case job execution time
v_i	Burstiness of the execution demand
R_i	Long-term arrival rate of T_i's jobs
B_i	Burstiness of the arrival curve
u_i	T_i's long-term utilization
U_{sum}	Total utilization
Θ_i below can be checked using the test in Section 5.4	
Θ_i	T_i's response-time bound
Output calculated using the input and $\{\Theta_i\}$	
$\alpha_i^{u\prime}(\Delta)\ (\alpha_i^{l\prime}(\Delta))$	Max. (min.) number of job completions of T_i over Δ
$\mathcal{B}'(\Delta)$	Min. guaranteed unused processor supply over Δ

5.1 Task Model

In this chapter, we consider a streaming task set $\tau = \{T_1, \ldots, T_n\}$ (see Section 1.6). Each task has incoming jobs that are processed by a multiprocessor defined as in Section 1.4. We also assume that all time quantities except the interval length Δ are integral.

As in prior work on real-time calculus, we wish to be able to accommodate very general assumptions concerning job executions and arrivals and the available service. Most of the remaining definitions in this section are devoted to formalizing the assumptions we require. Table 5.1 summarizes the notation introduced in this section.

Definition 5.1. $\gamma_i^u(k)\ (\gamma_i^l(k))$ denotes an upper (lower) bound on the total execution time of any k consecutive jobs of T_i. (We assume $\gamma_i^u(k) = \gamma_i^l(k) = 0$ for all $k \leq 0$ and $\gamma_i^u(k) \leq \gamma_i^u(k+1)$ and $\gamma_i^l(k) \leq \gamma_i^l(k+1)$.) These definitions are equivalent to the workload demand curves in (Maxiaguine, 2005).

Example 5.1. Suppose that task T_i's job execution times follow a pattern $1, 5, 2, 1, 5, 2, \ldots$. Then, $\gamma_i^u(1) = 5$, $\gamma_i^u(2) = 7$, $\gamma_i^u(3) = 8$, $\gamma_i^u(4) = 13$, etc. Also, $\gamma_i^l(1) = 1$, $\gamma_i^l(2) = 3$, $\gamma_i^l(3) = 8$, $\gamma_i^l(4) = 9$, etc.

Definition 5.2. The *arrival function* $\alpha_i^u(\Delta)\ (\alpha_i^l(\Delta))$ provides an upper (lower) bound on the number

of jobs of T_i that can arrive within *any* time interval $(x, x + \Delta]$, where $x \geq 0$ and $\Delta > 0$ (Chakraborty et al., 2006). (We assume $\alpha_i^u(\Delta) = 0$ for all $\Delta \leq 0$.) $\alpha_i(\Delta)$ denotes the pair $(\alpha_i^u(\Delta), \alpha_i^l(\Delta))$.

Example 5.2. The widely-studied periodic and sporadic task models are subcases of this more general task model. In both models, each job of T_i requires at most $e_i^{\max}$ execution units and consecutive job arrivals are separated by at least p_i time, where p_i is the *period* of T_i. Therefore, for both models, $\alpha_i^u(\Delta) = \left\lceil \frac{\Delta}{p_i} \right\rceil$ and $\gamma_i^u(k) = k \cdot e_i^{\max}$.

Definition 5.3. Let $\mathcal{A}_i^{-1}(k) = \inf\{\Delta \mid \alpha_i^u(\Delta) > k\}$, where $\Delta > 0$. This function characterizes the minimum length of the time interval $(x, x + \Delta]$ during which jobs $T_{i,j+1}, \ldots, T_{i,j+k}$ can be released for some j, assuming $T_{i,j}$ is released at time x. We define $\mathcal{A}_i^{-1}(0) = 0$ and require that there exists $K_i \geq 1$ such that

$$\mathcal{A}_i^{-1}(K_i) \geq \gamma_i^u(K_i). \tag{5.1}$$

We further require that there exists $R_i > 0$ and $B_i \geq 0$, where $R_i = \lim_{\Delta \to +\infty} \frac{\alpha_i^u(\Delta)}{\Delta}$, such that

$$\alpha_i^u(\Delta) \leq R_i \cdot \Delta + B_i \text{ for all } \Delta \geq 0. \tag{5.2}$$

Also, we assume that there exists $\overline{e_i} > 0$ and $v_i \geq 0$, where $\overline{e_i} = \lim_{k \to +\infty} \frac{\gamma_i^u(k)}{k}$, such that

$$\gamma_i^u(k) \leq \overline{e_i} \cdot k + v_i \text{ for all } k \geq 1. \tag{5.3}$$

(5.1) is needed in order to prevent task T_i from overloading the system. In (5.2), R_i characterizes the long-term arrival rate of task T_i's jobs and B_i characterizes the degree of burstiness of the arrival sequence. In (5.3), the parameter $\overline{e_i}$ denotes the average worst-case job execution time of T_i.

Definition 5.4. Let $u_i = R_i \cdot \overline{e_i}$. This quantity denotes the average long-term utilization of task T_i. We require that $0 < u_i \leq 1$. Let $U_{sum} = \sum_{T_i \in \tau} u_i$.

Example 5.3. Under the sporadic task model, $R_i = \lim_{\Delta \to +\infty} \left(\left\lfloor \frac{\Delta}{p_i} \right\rfloor + 1 \right) / \Delta = \frac{1}{p_i}$ and $\overline{e_i} = e_i^{\max}$, so $u_i = R_i \cdot \overline{e_i} = \frac{e_i^{\max}}{p_i}$.

Definition 5.5. Let $\mathsf{supply}_h(t, \Delta)$ be the total amount of processor time available to tasks in τ on processor h in the interval $[t, t + \Delta)$, where $\Delta \geq 0$. Let $\mathsf{Supply}(t, \Delta) = \sum_{h=1}^m \mathsf{supply}_h(t, \Delta)$ be the cumulative processor supply in the interval $[t, t + \Delta)$.

Though we desire to make our analysis compatible with the real-time calculus framework, which requires that individual processor supplies be known, there exist many settings in which individual processor supply functions are not known and a lower bound on the cumulative available processor time is provided instead. (In uniprocessor real-time calculus, the available service is described as the number of incoming events processed by a PE during a time interval.) Note that if individual processor supply guarantees are known, a lower bound on the cumulative guaranteed supply can be computed easily.

Definition 5.6. Let $\mathcal{B}(\Delta) \leq \mathsf{Supply}(t, \Delta)$ be the guaranteed total time that all processors can provide to the tasks in τ during any time interval $[t, t + \Delta)$, where $\Delta \geq 0$. We assume that

$$\mathcal{B}(\Delta) \geq \max(0, \widehat{U} \cdot (\Delta - \sigma_{tot})), \tag{5.4}$$

where $\widehat{U} \in (0, m]$ and $\sigma_{tot} \geq 0$. We let F be the number of processors that are always available at any time. If all processors have unit speed (as we have assumed), then $F = \max\{y \mid \forall \Delta \geq 0 :: \mathcal{B}(\Delta) \geq y \cdot \Delta\}$.

In the above definition, the parameters $\widehat{U}$, which is the total long-term fraction of processor time available to the tasks in τ on the entire platform, and σ_{tot}, which is the maximum duration of time when all processors are unavailable, are similar to those in the bounded delay model (Mok et al., 2001).

Example 5.4. Consider the system from Example 1.2 in Section 1.4. The availability pattern for one processor, which repeats every eight time units, is shown in Figure 5.2(a); intervals of unavailability are shown as shaded regions. For processor 1, the minimum amount of time that is guaranteed to real-time tasks over any interval of length Δ is zero if $\Delta \leq 2$, $\Delta - 2$ if $2 \leq \Delta \leq 4$, and so on. Figure 5.2(b) shows the minimum amount of time $\mathcal{B}(\Delta)$ that is available on processor 1 for tasks over any interval $[t, t + \Delta]$. It also shows a lower bound $\max(0, \widehat{U}(\Delta - \sigma_{tot}))$, where $\widehat{U} = \frac{5}{8}$ and $\sigma_{tot} = 2$, which bounds $\mathcal{B}(\Delta)$ from below.

We require that (5.5) below holds for otherwise the system is overloaded and job response times could be unbounded. This inequality is analogous to the utilization constraint in (3.1) in Section 3.1.

$$U_{sum} \leq \widehat{U} \tag{5.5}$$

We assume that released jobs are placed into a single global ready queue. When choosing a new job to schedule, the scheduler selects (and dequeues) the ready job of highest priority. An unfinished job is
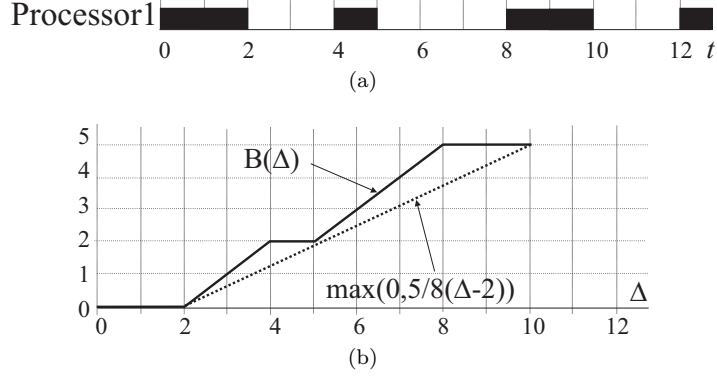
Figure 5.2: **(a)** Unavailable time instants and **(b)** service function in Example 5.4.

pending if it is released. A pending job is *ready* if its predecessor (if any) has completed execution. Note that, the jobs of each task execute sequentially. Job priorities are determined according to the following definition, which is a specialization of Definition 3.1 (see Section 3.1).

Definition 5.7. (prioritization rules) Associated with each job $T_{i,j}$ is a constant value $\chi_{i,j}$. If $\chi_{i,j} < \chi_{k,h}$ or $\chi_{i,j} = \chi_{k,h} \wedge (i < k \vee (i = k \wedge j < h))$, then the priority of $T_{i,j}$ is higher than that of $T_{k,h}$, denoted $T_{i,j} \prec T_{k,h}$. Additionally, we assume $j < h$ implies $\chi_{i,j} \leq \chi_{i,h}$ for each task T_i.

Example 5.5. As shown in Section 3.2, global earliest-deadline-first (GEDF) priorities can be defined by setting $\chi_{i,j} = r_{i,j} + D_i$ for each job $T_{i,j}$, where D_i is T_i's relative deadline. Global first-in-first-out (FIFO) priorities can be defined by setting $\chi_{i,j} = r_{i,j}$, and static priorities can be defined by setting $\chi_{i,j}$ to a constant.

In this chapter, we study three problems. First, given a task set $\tau = \{T_1, \ldots, T_n\}$ and a multiprocessor platform characterized by a cumulative guaranteed processor time $\mathcal{B}(\Delta)$, we develop a sufficient test that verifies whether the maximum job response time of a task $T_i \in \tau$, $\max_j(f_{i,j} - r_{i,j})$, is at most Θ_i, where

$$\Theta_i \geq \max_{j \geq 1}(\gamma_i^u(j) - \mathcal{A}_i^{-1}(j-1)). \tag{5.6}$$

The right-hand side of (5.6) is the maximum job response time bound of T_i when it is scheduled on a dedicated processor. Consider a sequence of j consecutive jobs $T_{i,a}, \ldots, T_{i,a+j-1}$ scheduled on a dedicated processor such that $T_{i,a}$ starts its execution at $r_{i,a}$ and $r_{i,k} \leq f_{i,k-1}$ for $k \in [a+1, a+j-1]$. The response time of job $T_{i,a+j-1}$ is $f_{i,a+j-1} - r_{i,a+j-1} = (f_{i,a+j-1} - r_{i,a}) - (r_{i,a+j-1} - r_{i,a})$. Because the processor is dedicated and jobs execute back-to-back, $f_{i,a+j-1} - r_{i,a} \leq \gamma_i^u(j)$. Below, in Section 5.4 in Lemma 5.2, we show that $r_{i,a+j-1} - r_{i,a} \geq \mathcal{A}_i^{-1}(j-1)$. Thus, $f_{i,a+j-1} - r_{i,a+j-1} \leq \gamma_i^u(j) - \mathcal{A}_i^{-1}(j-1)$.

111

If Θ_i equals the relative deadline of a job, then the proposed test will check whether the system is hard-real-time schedulable. Alternatively, if deadlines are allowed to be missed and Θ_i includes the maximum allowed deadline tardiness, then the test will check soft-real-time schedulability. Such a test allows workloads to be considered that fundamentally require global scheduling approaches.

In settings where response-time bounds are not known, they must be determined. The second problem we consider is a derivation of closed-form expressions for calculating response-time bounds directly from task and supply parameters for a large class of scheduling algorithms. These response-time bounds can be directly used for calculating stream and supply outputs. It is also possible to refine the obtained response-time bounds by incrementally decreasing them and running the schedulability test to see if the smaller bounds are also valid.

Finally, given per-task bounds on maximum job response times, we characterize the sequence of job completion events for each task T_i by deriving the next-stage arrival functions $\alpha_i^{u'}(\Delta)$ and $\alpha_i^{l'}(\Delta)$, and the remaining processor supply $\mathcal{B}'(\Delta)$ (see Figure 1.9). These functions, in turn, can serve as inputs to subsequent PEs, thereby resulting in a compositional technique.

5.2 Calculating $\alpha_i^{u'}$ and $\alpha_i^{l'}$

Let $\alpha_i^{u'}(\Delta)$ $(\alpha_i^{l'}(\Delta))$ be the maximum (respectively, minimum) number of job completions of task T_i over an interval $(x, x+\Delta]$, where $x \geq 0$. Bounds on these functions can be computed using Theorem 5.1 below. The following definition is used in the statement of the theorem.

Definition 5.8. Let ${\gamma^l_i}^{-1}(\Delta) = \inf\{k \mid k \text{ is integral and } \gamma_i^l(k) \geq \Delta\}$, where $\Delta > 0$, be the pseudoinverse function of the lower bound on the execution time function $\gamma_i^l(k)$ (note that we use a non-strict inequality because $\gamma_i^l(k)$ is not defined for non-integral values of k). For $\Delta = 0$, we define ${\gamma^l_i}^{-1}(0) = 0$.

Example 5.6. The function ${\gamma^l_i}^{-1}(\Delta)$ gives an upper bound on the number of jobs that can complete over any interval $(x, x+\Delta]$, where $\Delta > 0$. For example, if $\Delta = \gamma_i^l(1)$, then at most one job can complete over any interval $(x, x + \gamma_i^l(1)]$. Similarly, if $\Delta = \gamma_i^l(k)$ for some k, then at most k jobs can complete over any interval $(x, x + \gamma_i^l(k)]$.

Theorem 5.1. *If the response time of any job of T_i is at most Θ_i, then $\alpha_i^{u'}(\Delta) \leq \min\left({\gamma^l_i}^{-1}(\Delta),\right.$ $\left.\alpha_i^u(\Delta + \Theta_i - \gamma_i^l(1))\right)$ and $\alpha_i^{l'}(\Delta) \geq \alpha_i^l(\Delta - \Theta_i + \gamma_i^l(1))$.*

Proof. We first prove the first inequality. Consider an interval $(t_1, t_2]$ such that at least one job of T_i completes within it and let $\Delta = t_2 - t_1$. Let N_1, (N_2) be the index of the first (last) job of T_i completed

within $(t_1, t_2]$. Then,

$$f_{i,N_1} > t_1 \quad \text{and} \quad f_{i,N_2} \leq t_2. \tag{5.7}$$

By the condition of the theorem, job $T_{i,j}$'s response time $f_{i,j} - r_{i,j}$ is at most Θ_i. By the definition of response time and Definition 5.1, $f_{i,j} - r_{i,j}$ is at least $\gamma_i^l(1)$. From (5.7), we thus have $r_{i,N_1} > t_1 - \Theta_i$ and $r_{i,N_2} \leq t_2 - \gamma_i^l(1)$. Thus, the number of jobs completed within the interval $(t_1, t_2]$, $N_2 - N_1 + 1$, is at most the number of jobs released within the interval $(t_1 - \Theta_i, t_2 - \gamma_i^l(1)]$. By Definition 5.2, we have $N_2 - N_1 + 1 \leq \alpha_i^u(t_2 - \gamma_i^l(1) - t_1 + \Theta_i) = \alpha_i^u(\Delta + \Theta_i - \gamma_i^l(1))$. Moreover, from Definition 5.8, it follows that at most $\gamma_i^{l^{-1}}(\Delta)$ jobs can complete within an interval of length $\Delta > 0$.

We now prove the second inequality. Consider an interval $(t_1, t_2]$ and let $\Delta = t_2 - t_1$. Let N_1, (N_2) be the index of the last (respectively, first) job of T_i completed at or before time t_1 (respectively, after time t_2). Then,

$$f_{i,N_1} \leq t_1 \quad \text{and} \quad f_{i,N_2} > t_2. \tag{5.8}$$

By the condition of the theorem, job $T_{i,j}$'s response time $f_{i,j} - r_{i,j}$ is at most Θ_i. By the definition of response time and Definition 5.1, $f_{i,j} - r_{i,j}$ is at least $\gamma_i^l(1)$. From (5.8), we thus have $r_{i,N_2} > t_2 - \Theta_i$ and $r_{i,N_1} \leq t_1 - \gamma_i^l(1)$. Thus, the number of jobs completed within the interval $(t_1, t_2]$, $N_2 - N_1 - 1$, is at least the number of jobs released within the interval $(t_1 - \gamma_i^l(1), t_2 - \Theta_i]$. By Definition 5.2, we have $N_2 - N_1 - 1 \geq \alpha_i^l(t_2 - \Theta_i - t_1 + \gamma_i^l(1)) = \alpha_i^l(\Delta - \Theta_i + \gamma_i^l(1))$. $\qquad \square$

5.3 Calculating $\mathcal{B}'(\Delta)$

We now calculate a lower bound $\mathcal{B}'(\Delta)$ on processor time that is available after scheduling tasks $T_1, \ldots, T_n$. We first upper-bound the total allocation of jobs of T_i over any interval of length Δ.

Definition 5.9. Let $\mathsf{A}(T_{i,y}, I)$ (respectively, $\mathsf{A}(T_i, I)$) be the amount of time for which job $T_{i,y}$ (respectively, task T_i) executes within the set of intervals I.

Lemma 5.1. *If the response time of any job of T_i is at most Θ_i, then $\mathsf{A}(T_i, [t, t+\Delta)) \leq \min(\Delta, \gamma_i^u(\alpha_i^u(\Delta + \Theta_i)))$.*

Proof. Consider an interval $[t, t+\Delta)$. The condition of the lemma implies that all of T_i's jobs released at or before time $t - \Theta_i$ complete by time t. Thus, the allocation of T_i within $[t, t + \Delta)$, $\mathsf{A}(T_i, [t, t + \Delta))$, is upper-bounded by the maximum execution demand of T_i's jobs released within the interval $(t - \Theta_i, t + \Delta]$. By Definition 5.2, there are at most $\alpha_i^u(\Delta + \Theta_i)$ such jobs, and by Definition 5.1, their total execution

113

demand is at most $\gamma_i^u(\alpha_i^u(\Delta+\Theta_i))$. We thus have $\mathsf{A}(T_i,[t,t+\Delta)) \leq \gamma_i^u(\alpha_i^u(\Delta+\Theta_i))$. Also, $\mathsf{A}(T_i,[t,t+\Delta))$ cannot exceed the length of the interval $[t,t+\Delta)$. $\qquad\square$

Theorem 5.2. *If, for each task T_i, the response time of any of its jobs is at most Θ_i, then at least*

$$\mathcal{B}'(\Delta) = \sup_{0 \leq y \leq \Delta} (Z(y)) \tag{5.9}$$

time units are available over any interval of length $\Delta \geq 0$, where $Z(y) = \max\left(0, \mathcal{B}(y) - \sum_{T_i \in \tau} \min(y, \gamma_i^u(\alpha_i^u(y+$ $\Theta_i)))\right)$. Additionally, (5.4) for $\mathcal{B}'(\Delta)$ holds with $\widehat{U}' = \widehat{U} - U_{sum}$ and $\sigma'_{tot} = (\widehat{U} \cdot \sigma_{tot} + \sum_{T_i \in \tau}(u_i \cdot \Theta_i +$ $\overline{e_i} \cdot B_i + v_i))/\widehat{U}'$.

Proof. Consider an interval $[t, t+y)$, where $y \leq \Delta$. Let $\mathsf{Supply}'(t,y)$ be the amount of supply that is available after scheduling the tasks in τ in this interval. By Definitions 5.5 and 5.9, we have

$$\mathsf{Supply}'(t,y) = \mathsf{Supply}(t,y) - \sum_{T_i \in \tau} \mathsf{A}(T_i, [t, t+y))$$

$$\{\text{by Definition 5.6}\}$$

$$\geq \max\left(0, \mathcal{B}(y) - \sum_{T_i \in \tau} \mathsf{A}(T_i, [t, t+y))\right)$$

$$\{\text{by Lemma 5.1}\}$$

$$\geq \max\left(0, \mathcal{B}(y) - \sum_{T_i \in \tau} \min(y, \gamma_i^u(\alpha_i^u(y+\Theta_i)))\right)$$

$$\{\text{by the definition of } Z(y) \text{ in the statement of the theorem}\}$$

$$= Z(y). \tag{5.10}$$

Additionally, $\mathsf{Supply}'(t,\Delta) \geq \sup_{0 \leq y \leq \Delta}(\mathsf{Supply}'(t,y))$. From this inequality and (5.10), we have $\mathsf{Supply}'(t,\Delta) \geq \sup_{0 \leq y \leq \Delta}(Z(y)) = \mathcal{B}'(\Delta)$.

We are left with finding coefficients $\widehat{U}'$ and σ'_{tot} such that (5.4) holds for $\mathcal{B}'(\Delta)$. Setting (5.4) (for $\mathcal{B}(\Delta)$) into the definition of $Z(y)$, we have

$$Z(y) \geq \max\left(0, \max(0, \widehat{U} \cdot (y-\sigma_{tot})) - \sum_{T_i \in \tau} \min(y, \gamma_i^u(\alpha_i^u(y+\Theta_i)))\right)$$

$$\geq \max\left(0, \widehat{U} \cdot (y-\sigma_{tot}) - \sum_{T_i \in \tau} \min(y, \gamma_i^u(\alpha_i^u(y+\Theta_i)))\right)$$

$$\{\text{by (5.2) and (5.3)}\}$$

114

$$\geq \max\left(0, \widehat{U} \cdot (y - \sigma_{tot}) - \sum_{T_i \in \tau} (\overline{e_i} \cdot (R_i \cdot (y + \Theta_i) + B_i) + v_i)\right)$$

{by Definition 5.4}

$$= \max\left(0, \widehat{U} \cdot (y - \sigma_{tot}) - \sum_{T_i \in \tau} (u_i \cdot y + u_i \cdot \Theta_i + \overline{e_i} \cdot B_i + v_i)\right)$$

$$= \max\left(0, \widehat{U} \cdot (y - \sigma_{tot}) - U_{sum} \cdot y + \sum_{T_i \in \tau} (u_i \cdot \Theta_i + \overline{e_i} \cdot B_i + v_i)\right)$$

$$= \max\left(0, (\widehat{U} - U_{sum}) \cdot y - \widehat{U} \cdot \sigma_{tot} - \sum_{T_i \in \tau} (u_i \cdot \Theta_i + \overline{e_i} \cdot B_i + v_i)\right)$$

{by the definition of $\widehat{U}'$ and σ_{tot}' in the statement of the theorem}

$$= \max\left(0, \widehat{U}' \cdot (y - \sigma_{tot}')\right). \tag{5.11}$$

Finally, by (5.9) and (5.11), $\mathcal{B}'(\Delta) \geq \sup_{0 \leq y \leq \Delta} \left(\max\left(0, \widehat{U}' \cdot (y - \sigma_{tot}')\right)\right) = \max\left(0, \widehat{U}' \cdot (\Delta - \sigma_{tot}')\right)$. Thus, (5.4) holds with $\mathcal{B}'(\Delta)$, $\widehat{U}'$, and σ_{tot}' as defined. ◻

5.4   Multiprocessor Schedulability Test

In this section, we present the core analysis of our framework in the form of a schedulability test (given in Corollary 5.1 later in this section) that checks whether a pre-defined response-time bound Θ_i is not violated for a task T_i.

As noted earlier, the way jobs are prioritized according to Definition 5.7 is similar to GEDF or static priority scheduling. In this chapter, we extend techniques from (Baruah, 2007) (the "SB-test" in Section 2.2.1) and (Leontyev and Anderson, 2008b) in order to incorporate more general job arrivals and execution models.

Similarly to (Devi, 2006), we derive our test by ordering jobs by their priorities and assuming that $T_{\ell,q}$ is the first job for which $f_{\ell,q} > r_{\ell,q} + \Theta_\ell$. We further assume that, for each job $T_{a,b}$ such that $T_{a,b} \prec T_{\ell,q}$,

$$f_{a,b} \leq r_{a,b} + \Theta_a. \tag{5.12}$$

We first derive a necessary condition for $T_{\ell,q}$ to violate its response-time bound by considering an interval that includes the time when $T_{\ell,q}$ becomes ready and the latest time when $T_{\ell,q}$ is allowed to complete, which is $r_{\ell,q} + \Theta_\ell$. This interval is parametrized by a number $k \in [1, K_\ell]$ (see Definition 5.3) and δ (defined later in this section), which determine its length, $\delta + \Theta_\ell$. (The range of δ depends on k and ℓ.) In essence, the parameter k defines the number of $T_{\ell,q}$'s predecessors (including $T_{\ell,q}$ itself)

that complete "too late" to warrant $T_{\ell,q}$'s timely completion in the presence of other tasks. During this interval, we consider demand due to competing higher-priority jobs that can interfere with $T_{\ell,q}$ or its predecessors. We then perform the following three steps:

S1: Compute the minimum guaranteed supply $\mathcal{B}(\delta + \Theta_\ell)$ over the interval of interest.

S2: Given a finite upper bound $M_\ell^*(\delta, \tau, m)$ on the competing demand and a finite upper bound on the unfinished work due to job $T_{\ell,q}$ and its predecessors, $E_\ell^*(k)$, define a sufficient test for checking whether T_ℓ's response-time bound is not violated by checking that $M_\ell^*(\delta, \tau, m) + (m-1) \cdot (E_\ell^*(k) - 1) < \mathcal{B}(\delta + \Theta_\ell)$ holds for each $k \in [1, K_\ell]$ and δ defined with respect to k and ℓ.

S3: Calculate $M_\ell^*(\delta, \tau, m)$ and $E_\ell^*(k)$ as used in **S2**.

5.4.1 Steps S1 and S2

To avoid distracting "boundary cases," we henceforth assume that the schedule being analyzed is prepended with a schedule in which response-time bounds are not violated that is long enough to ensure that all predecessor jobs referenced in the proof exist. We begin with the following definition.

Definition 5.10. Let $\alpha_i^+(\Delta) = \lim_{\epsilon \to +0} \alpha_i^u(\Delta + \epsilon)$. This function provides an upper bound on the number of jobs released within any interval $[x, x + \Delta]$, where $x \geq 0$ and $\Delta \geq 0$. (We assume $\alpha_i^+(\Delta) = 0$ for all $\Delta < 0$.)

The next example illustrates the difference between the functions α_i^u and α_i^+.

Example 5.7. Consider a task T_i, whose jobs arrive periodically with period p_i. The maximum number of jobs that can arrive within an interval $(x, x + 2 \cdot p_i]$ is thus $\alpha_i^u(2 \cdot p_i) = \left\lceil \frac{2 \cdot p_i}{p_i} \right\rceil = 2$. However, the maximum number of jobs that can arrive within the interval $[x, x + 2 \cdot p_i]$ is $\alpha_i^+(2 \cdot p_i) = \lim_{\epsilon \to +0} \alpha_i^u(2 \cdot p_i + \epsilon) = 3$. In general, under the sporadic task model, $\alpha_i^+(\Delta) = \left\lfloor \frac{\Delta}{p_i} \right\rfloor + 1$.

We start the derivation by proving a lemma and several claims. The following lemma specifies the minimum time between the arrivals of jobs $T_{h,g}$ and $T_{h,g-i}$.

Lemma 5.2. $r_{h,g} - r_{h,g-i} \geq \mathcal{A}_h^{-1}(i)$.

Proof. Let $\Delta' = r_{h,g} - r_{h,g-i}$. Let

$$\Delta^* = \inf\{\Delta \mid \alpha_h^+(\Delta) \geq i + 1\}. \tag{5.13}$$

Because jobs $T_{h,g-i}, \ldots, T_{h,g}$ are released within the interval $[r_{h,g-i}, r_{h,g}]$, by Definition 5.10, $\alpha_h^+(\Delta') \geq i + 1$. Therefore, by (5.13),

$$r_{h,g} - r_{h,g-i} = \Delta' \geq \Delta^*. \tag{5.14}$$

We now consider two cases.

Case 1: $\alpha_h^u(\Delta^*) > i$. In this case, $\Delta^* \overset{\{\text{by Definition 5.10}\}}{\geq} \inf\{\Delta \mid \alpha_h^u(\Delta) > i\} \overset{\{\text{by Definition 5.3}\}}{=} \mathcal{A}_h^{-1}(i)$. The lemma follows from this and (5.14).

Case 2: $\alpha_h^u(\Delta^*) \leq i$. Because $\alpha_h^u(\Delta)$ is non-decreasing, $\alpha_h^u(\Delta^*) \leq i$ implies

$$\alpha_\ell^u(\Delta) \leq i \text{ for each } \Delta \leq \Delta^*. \tag{5.15}$$

Further, by (5.13),

$$\alpha_h^+(\Delta) < i + 1, \text{for each } \Delta < \Delta^*. \tag{5.16}$$

Suppose that for some $\Delta'' > \Delta^*$, $\alpha_h^u(\Delta'') \leq i$. Because $\alpha_h^u(\Delta)$ is non-decreasing, this implies $\alpha_h^u(\Delta_x) \leq i$ for each $\Delta_x \in [\Delta^*, \Delta'']$. The latter implies $\alpha_h^+(\Delta_x) = \lim_{\epsilon \to +0} \alpha_h^u(\Delta_x + \epsilon) \leq i$ for each $\Delta_x \in [\Delta^*, \Delta'']$. From this and (5.16), we have $\alpha_h^+(\Delta) < i + 1$ for each $\Delta < \Delta''$. Since $\Delta'' > \Delta^*$, we have a contradiction to (5.13). Therefore, $\alpha_h^u(\Delta) > i$ for each $\Delta > \Delta^*$. From this and (5.15), we have $\Delta^* = \inf\{\Delta \mid \alpha_h^u(\Delta) > i\} \overset{\{\text{by Definition 5.3}\}}{=} \mathcal{A}_h^{-1}(i)$. The lemma follows from this equality and (5.14). $\qquad\square$

The next two claims establish a lower bound on the maximum job response time and an upper bound on the finish times of certain jobs that can be used in addition to (5.12).

Claim 5.1: $\Theta_i \geq \gamma_i^u(1)$.

Proof. By (5.6), $\Theta_i \geq \max_{j \geq 1}(\gamma_i^u(j) - \mathcal{A}_i^{-1}(j-1)) \geq \gamma_i^u(1) - \mathcal{A}_i^{-1}(0)$. By Definition 5.3, $\mathcal{A}_i^{-1}(0) = 0$. $\quad\square$

Claim 5.2: $f_{\ell,q-K_\ell} \leq r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(K_\ell)$.

Proof. By (5.12), for $i \geq 1$,

$$f_{\ell,q-i} \leq r_{\ell,q-i} + \Theta_\ell$$

$$= r_{\ell,q-i} - r_{\ell,q} + r_{\ell,q} + \Theta_\ell$$

$$\{\text{by Lemma 5.2}\}$$

$$\leq r_{\ell,q} + \Theta_\ell - \mathcal{A}_\ell^{-1}(i). \tag{5.17}$$

By (5.1), $-\mathcal{A}_\ell^{-1}(K_\ell) \leq -\gamma_\ell^u(K_\ell)$. Setting this and $i = K_\ell$ into (5.17), we get the required result. $\square$

Job $T_{\ell,q}$ can violate its response-time bound for the following reasons. If $T_{\ell,q-1}$ completes by time $r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(1)$, then $T_{\ell,q}$ may finish its execution after $r_{\ell,q} + \Theta_\ell$ if, after time $\max(f_{\ell,q-1}, r_{\ell,q})$, higher-priority jobs deprive it of processor time or one or more processors are unavailable. Alternatively, $T_{\ell,q-1}$ may complete *after* time $r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(1)$, which can happen if the minimum job inter-arrival time for T_ℓ is less than $\gamma_\ell^u(1)$. In this situation, $T_{\ell,q}$ could violate its response-time bound even if it executes uninterruptedly within $[f_{\ell,q-1}, r_{\ell,q} + \Theta_\ell)$. In this case, T_ℓ's response-time bound is violated because $T_{\ell,q-1}$ completes "late," namely after time $r_{\ell,q}$ (recall that, by Claim 5.1, $\Theta_\ell \geq \gamma_\ell^u(1)$). However, this implies that T_ℓ is pending continuously throughout the interval $[r_{\ell,q-1}, r_{\ell,q} + \Theta_\ell)$, and hence, we can examine the execution of jobs $T_{\ell,q-1}$ and $T_{\ell,q}$ together. In this case, we need to consider the completion time of job $T_{\ell,q-2}$. If $f_{\ell,q-2} \leq r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(2)$, then job $T_{\ell,q}$ may exceed its response-time bound if this job and its predecessor, $T_{\ell,q-1}$, experience interference from higher-priority jobs or some processors are unavailable during the time interval $[\max(f_{\ell,q-2}, r_{\ell,q-1}), r_{\ell,q} + \Theta_\ell)$. On the other hand, if $f_{\ell,q-2} > r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(2)$, then $T_{\ell,q}$ can complete after time $r_{\ell,q} + \Theta_\ell$ even if T_ℓ executes uninterruptedly within $[f_{\ell,q-2}, r_{\ell,q} + \Theta_\ell)$. Continuing by considering predecessor jobs $T_{\ell,q-k}$ in this manner, we will exhaust all possible reasons for the response-time bound violation. Note that it is sufficient to consider only jobs $T_{\ell,q-1}, \ldots, T_{\ell,q-K_\ell}$ since, by Claim 5.2, $f_{\ell,q-K_\ell} \leq r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(K_\ell)$. Assuming that, for job $T_{\ell,q-k}$, $f_{\ell,q-k} \leq r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(k)$, we define the *problem window* for jobs $T_{\ell,q-k+1}, \ldots, T_{\ell,q}$ as $[r_{\ell,q-k+1}, r_{\ell,q} + \Theta_\ell)$. (This problem window definition is a significant difference when comparing our analysis to prior analysis pertaining to periodic or sporadic systems.)

Definition 5.11. Let $\lambda \in [1, K_\ell]$ be the smallest integer such that $f_{\ell,q-\lambda} \leq r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda)$. By Claim 5.2, such a λ exists.

Claim 5.3. T_ℓ *is ready (i.e., has a ready job) at each instant of the interval* $[r_{\ell,q-k+1}, r_{\ell,q} + \Theta_\ell)$ *for each* $k \in [1, \lambda]$.

Proof. To prove the claim, we first show that, for each $k \in [1, \lambda]$, T_ℓ is ready continuously within $[r_{\ell,q-k+1}, f_{\ell,q})$. Because T_ℓ is ready within the interval $[r_{\ell,q}, f_{\ell,q})$, this is true for $k = 1$. If $k > 1$ (in which case $\lambda > 1$), then $f_{\ell,q-j} > r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(j)$ for each $j \in [1, \lambda)$, by the selection of λ. From this,

118

we have

$$f_{\ell,q-j} > r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(j)$$

$$\{\text{because, by (5.6)}, \Theta_\ell \geq \gamma_\ell^u(j) - \mathcal{A}_\ell^{-1}(j-1)\}$$

$$\geq r_{\ell,q} - \mathcal{A}_\ell^{-1}(j-1)$$

$$\{\text{by Lemma 5.2}\}$$

$$\geq r_{\ell,q-j+1}.$$

Thus, the intervals $[r_{\ell,q-j}, f_{\ell,q-j})$ and $[r_{\ell,q-j+1}, f_{\ell,q-j+1})$, where consecutive jobs of T_ℓ are ready, overlap. Therefore, T_ℓ is ready continuously within $[r_{\ell,q-j}, f_{\ell,q})$ for each $j \in [1, \lambda)$, and hence, T_ℓ is ready continuously within $[r_{\ell,q-k+1}, f_{\ell,q})$ for each $k \in [2, \lambda]$. The claim follows from $[r_{\ell,q-k+1}, r_{\ell,q} + \Theta_\ell) \subset [r_{\ell,q-k+1}, f_{\ell,q})$; to see this, note that $f_{\ell,q} > r_{\ell,q} + \Theta_\ell$ holds, since $T_{\ell,q}$ violates its response-time bound. $\square$

Because $T_{\ell,q}$ violates its response-time bound, after time $r_{\ell,q-k+1}$, there are other higher-priority jobs that deprive T_ℓ of processor time or one or more processors are unavailable.

Definition 5.12. Let $\tau_p(t) = \{T_h \mid \text{for some } y,\ T_{h,y} \text{ is ready at time } t \text{ and } T_{h,y} \preceq T_{\ell,q}\}$. (The subscript p denotes the fact that these jobs have higher or equal priority.)

To indicate an excessive number of tasks with ready jobs of equal or higher priority at time t we will use the following predicate.

$$\text{IS\_HP}(t) = (|\tau_p(t)| \geq m \text{ or fewer than } |\tau_p(t)| \text{ tasks from } \tau_p(t) \text{ execute at time } t). \tag{5.18}$$

Definition 5.13. Let $t_0(k) \leq r_{\ell,q-k+1}$ be the earliest instant such that $\forall t \in [t_0(k), r_{\ell,q-k+1})$, $\text{IS\_HP}(t)$ holds. If such an instant does not exist, then let $t_0(k) = r_{\ell,q-k+1}$.

The definition below defines the jobs that can compete with $T_{\ell,q}$ or its predecessors.

Definition 5.14. Let $\mathcal{J}$ be the set of jobs $T_{i,y}$ such that **(i)** $T_{i,y} \preceq T_{\ell,q}$ or **(ii)** $T_{i,y} \succ T_{\ell,q}$, $i \neq \ell$, and $T_{i,y}$ executes at some time $t \in [t_0(k), r_{\ell,q} + \Theta_\ell)$ and $\text{IS\_HP}(t)$ holds. (More informally, $\mathcal{J}$ includes higher-or-equal-priority jobs and lower-priority jobs that cause non-preemptive blocking.) Note that $\mathcal{J}$ does not contain $T_{\ell,q}$'s successors.

In this chapter, we are mainly concerned with fully preemptive scheduling (i.e., $\mathcal{J}$ contains only higher-or-equal-priority jobs). However, the introduced definitions are constructed to support both fully
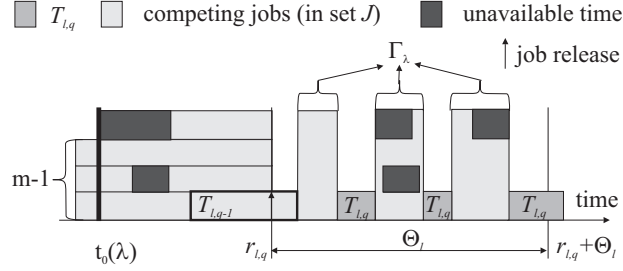
Figure 5.3: Conditions for a response-time bound violation for $\lambda = 1$.

preemptive and non-preemptive execution. Unless stated otherwise, we do not distinguish between the preemptive and non-preemptive cases. However, if non-preemptive execution is assumed, then we assume that all processors are always available. We discuss other differences in the analysis of the non-preemptive case in Section 5.4.3 and leave the consideration of non-preemptivity in the face of partial availability to future work.

Definition 5.13 generalizes the well-known concept of an *idle instant* in uniprocessor scheduling with respect to jobs in $\mathcal{J}$, as illustrated in Figure 5.3, which shows the response-time bound violation for job $T_{\ell,q}$ assuming $\lambda = 1$.

Our schedulability test for task T_ℓ is based upon summing the demand of competing jobs as defined above in Definition 5.14 executing within the interval $[t_0(\lambda), r_{\ell,q} + \Theta_\ell)$, which has length $r_{\ell,q} - t_0(\lambda) + \Theta_\ell$, and the unavailable time within this interval (see Figure 5.3).

Definition 5.15. Let $\mathsf{A}_{\mathcal{J}}(T_i, I) = \sum_{T_{i,y} \in \mathcal{J}} \mathsf{A}(T_{i,y}, I)$ be the allocation of task T_i's jobs in $\mathcal{J}$ over a set of intervals I.

Definition 5.16. Let $\mathsf{Res}_h(I)$ be the amount of time that is not available on processor h at time instants in the set of intervals I.

Definition 5.17. We call a processor $\mathcal{J}$-*busy* at time t if it executes a job in $\mathcal{J}$ or is unavailable. The total time for which processors are $\mathcal{J}$-busy within a set of intervals I is called the $\mathcal{J}$-*allocation* for I and is defined as $\widehat{\mathsf{A}_{\mathcal{J}}}(I) = \sum_{T_i \in \tau} \mathsf{A}_{\mathcal{J}}(T_i, I) + \sum_{h=1}^{m} \mathsf{Res}_h(I)$.

The following definition is used to calculate $\widehat{\mathsf{A}_{\mathcal{J}}}([t_0(\lambda), r_{\ell,q} + \Theta_\ell))$.

Definition 5.18. Let $\Gamma_\lambda \subseteq [r_{\ell,q-\lambda+1}, r_{\ell,q} + \Theta_\ell)$ be the set of intervals where all processors are $\mathcal{J}$-busy as shown in Figure 5.3. Let $\overline{\Gamma_\lambda} = [r_{\ell,q-\lambda+1}, r_{\ell,q} + \Theta_\ell) \setminus \Gamma_\lambda$. We let $|\Gamma_\lambda|$ (respectively, $|\overline{\Gamma_\lambda}|$) denote the total length of the intervals in Γ_λ (respectively, $\overline{\Gamma_\lambda}$).

We next calculate the $\mathcal{J}$-allocations for Γ_λ, $[t_0(\lambda), r_{\ell,q-\lambda+1})$, and $\overline{\Gamma_\lambda}$. Because all processors are $\mathcal{J}$-busy within Γ_λ, $\widehat{\mathsf{A}_{\mathcal{J}}}(\Gamma_\lambda) = m \cdot |\Gamma_\lambda|$. We now consider the interval $[t_0(\lambda), r_{\ell,q-\lambda+1})$.

Claim 5.4. *All processors are $\mathcal{J}$-busy within $[t_0(\lambda), r_{\ell,q-\lambda+1})$; that is,*

$$\widehat{\mathsf{A}_{\mathcal{J}}}([t_0(\lambda), r_{\ell,q-\lambda+1})) = m \cdot (r_{\ell,q-\lambda+1} - t_0(\lambda)).$$

Proof. Suppose that a processor is not $\mathcal{J}$-busy at time $t' \in [t_0(\lambda), r_{\ell,q-\lambda+1})$. Then it is either available and idle or executes a job that is not in $\mathcal{J}$. If the processor is idle at time t', then, because the scheduler being analyzed is work-conserving, all tasks in $\tau_p(t)$ execute at time t' and thus $|\tau_p(t')| \leq m - 1$. Thus, by (5.18), IS_HP(t') is *false*, which violates Definition 5.13. Alternatively, if, at time t', the processor executes a job $T_{x,y} \notin \mathcal{J}$, then, by Definition 5.14, $T_{x,y} \succ T_{\ell,q}$ and, by (5.18), IS_HP(t') is *false*, which also violates Definition 5.13. The given expression for $\widehat{\mathsf{A}_{\mathcal{J}}}([t_0(\lambda), r_{\ell,q-\lambda+1}))$ follows. $\square$

Finally, we consider the interval set $\overline{\Gamma_\lambda}$.

Claim 5.5. *Task T_ℓ executes at each time $t \in \overline{\Gamma_\lambda}$, and hence, $\mathsf{A}_{\mathcal{J}}(T_\ell, \overline{\Gamma_\lambda}) = |\overline{\Gamma_\lambda}|$.*

Proof. Suppose to the contrary that T_ℓ does not execute at some time $t \in \overline{\Gamma_\lambda}$. By Definition 5.18, there exists processor P that is not $\mathcal{J}$-busy at time t. By Claim 5.3, task T_ℓ is ready at each time $t \in [r_{\ell,q-\lambda+1}, r_{\ell,q} + \Theta_\ell)$. If P is idle, then, because the scheduler is work-conserving, all tasks in $\tau_p(t)$, including T_ℓ execute at time t. If P executes job $T_{x,y} \notin \mathcal{J}$, then, by Definition 5.14 and (5.18), IS_HP(t) is *false*, which implies that all tasks in $\tau_p(t)$, including T_ℓ execute at time t. In either case, we have a contradiction. $\square$

Lemma 5.3: $\widehat{\mathsf{A}_{\mathcal{J}}}([t_0(\lambda), r_{\ell,q} + \Theta_\ell)) \geq m \cdot (r_{\ell,q-\lambda+1} - t_0(\lambda)) + m \cdot |\Gamma_\lambda| + |\overline{\Gamma_\lambda}|$.

Proof. We sum up the $\mathcal{J}$-allocations for intervals $[t_0(\lambda), r_{\ell,q-\lambda+1})$, Γ_λ, and $\overline{\Gamma_\lambda}$ (see Figure 5.3; note that $r_{\ell,q-\lambda+1} = r_{\ell,q}$ here).

$$
\begin{aligned}
&\widehat{\mathsf{A}_{\mathcal{J}}}([t_0(\lambda), r_{\ell,q} + \Theta_\ell)) \\
&= \widehat{\mathsf{A}_{\mathcal{J}}}([t_0(\lambda), r_{\ell,q-\lambda+1})) + \widehat{\mathsf{A}_{\mathcal{J}}}(\Gamma_\lambda) + \widehat{\mathsf{A}_{\mathcal{J}}}(\overline{\Gamma_\lambda}) \\
&\quad \{\text{by Claim 5.4 and Definition 5.18}\} \\
&= m \cdot (r_{\ell,q-\lambda+1} - t_0(\lambda)) + m \cdot |\Gamma_\lambda| + \widehat{\mathsf{A}_{\mathcal{J}}}(\overline{\Gamma_\lambda}) \\
&\quad \{\text{by Definition 5.17}\} \\
&\geq m \cdot (r_{\ell,q-\lambda+1} - t_0(\lambda)) + m \cdot |\Gamma_\lambda| + \mathsf{A}_{\mathcal{J}}(T_\ell, \overline{\Gamma_\lambda}) \\
&\quad \{\text{by Claim 5.5}\} \\
&= m \cdot (r_{\ell,q-\lambda+1} - t_0(\lambda)) + m \cdot |\Gamma_\lambda| + |\overline{\Gamma_\lambda}| \qquad\qquad \square
\end{aligned}
$$

The values of $|\Gamma_\lambda|$ and $|\overline{\Gamma_\lambda}|$ depend on the amount of competing work due to $T_{\ell,q}$'s predecessors (including $T_{\ell,q}$ itself), which is determined as follows.

Definition 5.19. Let $W(T_{i,y}, t)$ denote the remaining execution time for job $T_{i,y}$ (if any) at time t. Let $\mathsf{W}_{\mathcal{J}}(T_i, t) = \sum_{T_{i,y} \in \mathcal{J}} W(T_{i,y}, t)$. In Figure 5.3, $\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1})$ corresponds to the execution demand of job $T_{\ell,q}$ and the unfinished work of job $T_{\ell,q-1}$ at time $r_{\ell,q}$.

Claim 5.6. (Proved in an appendix.) $\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) \leq r_{\ell,q} + \Theta_\ell - r_{\ell,q-\lambda+1}$.

The following lemma establishes constraints on the total length of the intervals Γ_λ and $\overline{\Gamma_\lambda}$.

Lemma 5.4. *If the response-time bound for $T_{\ell,q}$ is violated (as we have assumed), then $|\Gamma_\lambda| = r_{\ell,q} + \Theta_\ell - r_{\ell,q-\lambda+1} - \mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) + 1 + \mu$, where $\mu \geq 0$. (Note that, by Claim 5.6, this implies that $|\Gamma_\lambda| > 0$). Additionally, $|\overline{\Gamma_\lambda}| = \mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1 - \mu$.*

Proof. Suppose, contrary to the statement of the lemma, that the response-time bound for $T_{\ell,q}$ is violated and $\mu < 0$, i.e.,

$$|\Gamma_\lambda| < r_{\ell,q} + \Theta_\ell - r_{\ell,q-\lambda+1} - \mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) + 1. \tag{5.19}$$

Under these conditions, the total length of the intervals in $\overline{\Gamma_\lambda}$, where at least one available processor is not $\mathcal{J}$-busy, is $r_{\ell,q} + \Theta_\ell - r_{\ell,q-\lambda+1} - |\Gamma_\lambda| \overset{\{\text{by } (5.19)\}}{>} \mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1$. Thus, this total length is at least $\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1})$, as time is integral. By Claim 5.5, job $T_{\ell,q}$ or one of its predecessors executes at each time $t \in \overline{\Gamma_\lambda}$. Thus, job $T_{\ell,q}$ completes by time $r_{\ell,q} + \Theta_\ell$, which is a contradiction. Hence, $\mu \geq 0$. $|\overline{\Gamma_\lambda}|$ can be found as $|\overline{\Gamma_\lambda}| = r_{\ell,q} + \Theta_\ell - r_{\ell,q-\lambda+1} - |\Gamma_\lambda| = \mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1 - \mu$. $\qquad\square$

In the statement of Theorem 5.3, which defines a schedulability condition, the functions defined below are used.

Definition 5.20. Let $E_\ell^*(k)$ be a finite function of k such that $\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) \leq E_\ell^*(\lambda)$.

Definition 5.21. Let $M_\ell^*(\delta, \tau, m)$ be a finite function of δ, τ, and m such that

$$\sum_{T_i \in \tau} \mathsf{A}_{\mathcal{J}}(T_i, [t_0(\lambda), r_{\ell,q} + \Theta_\ell)) \leq M_\ell^*(r_{\ell,q} - t_0(\lambda), \tau, m).$$

(As mentioned earlier at the beginning of Section 5.4, $M_\ell^*(\delta, \tau, m)$ and $E_\ell^*(k)$ are calculated in order to test whether the response-time bound of T_ℓ is not violated. Later, in Section 5.4.2, we explain how $M_\ell^*(\delta, \tau, m)$ and $E_\ell^*(k)$ are calculated.)

Definition 5.22. We require that there exists a constant $H_\ell \geq 0$ such that, for all $\delta \geq 0$,

$$M_\ell^*(\delta, \tau, m) \leq U_{sum} \cdot \delta + H_\ell. \tag{5.20}$$

This requirement is reasonable because the growth rate of the total demand over the interval of interest, which has length $\delta + \Theta_\ell$, cannot be larger than the total long-term utilization of the tasks in τ for large values of δ. This also allows us to upper-bound our test's computational complexity. Henceforth, we omit the last two arguments of M_ℓ^*.

Definition 5.23. Let $\delta_\ell^{\max}(k) = \left\lfloor (H_\ell + (m-1) \cdot (E_\ell^*(k) - 1) + \widehat{U} \cdot \sigma_{tot} - \Theta_\ell \cdot \widehat{U})/(\widehat{U} - U_{sum}) \right\rfloor$.

We next calculate an upper bound on $\mathsf{Res}_h([t_0(\lambda), r_{\ell,q} + \Theta_\ell))$. For processor h and the interval $[t_0(\lambda), r_{\ell,q} + \Theta_\ell)$, by Definition 5.5,

$$\mathsf{Res}_h([t_0(\lambda), r_{\ell,q} + \Theta_\ell)) = (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - \mathsf{supply}_h(t_0(\lambda), r_{\ell,q} - t_0(\lambda) + \Theta_\ell). \tag{5.21}$$

Summing (5.21) for all h, we have

$$\sum_{h=1}^{m} \mathsf{Res}_h([t_0(\lambda), r_{\ell,q} + \Theta_\ell))$$

$$= \sum_{h=1}^{m} \left((r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - \mathsf{supply}_h(t_0(\lambda), r_{\ell,q} - t_0(\lambda) + \Theta_\ell) \right)$$

$$\{\text{by Definition 5.5}\}$$

$$= m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - \mathsf{Supply}(t_0(\lambda), r_{\ell,q} - t_0(\lambda) + \Theta_\ell)$$

$$\{\text{by Definition 5.6}\}$$

$$\leq m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - \mathcal{B}(r_{\ell,q} - t_0(\lambda) + \Theta_\ell). \tag{5.22}$$

The following theorem will be used to define our schedulability test.

Theorem 5.3. *If the response-time bound Θ_ℓ is violated for $T_{\ell,q}$ (as we have assumed), then, for $k = \lambda$ and some $\delta \in [\mathcal{A}_\ell^{-1}(\lambda - 1), \delta_\ell^{\max}(\lambda)]$ (such that $\delta = r_{\ell,q} - t_0(\lambda)$),*

$$M_\ell^*(\delta) + (m-1) \cdot (E_\ell^*(k) - 1) \geq \mathcal{B}(\delta + \Theta_\ell). \tag{5.23}$$

Proof. Consider job $T_{\ell,q}$, $k = \lambda$, and time instants $r_{\ell,q-\lambda+1}$ and $t_0(\lambda)$ as defined in Definitions 5.11

123

and 5.13. To establish (5.23), we consider the total $\mathcal{J}$-allocation within the interval $[t_0(\lambda), r_{\ell,q} + \Theta_\ell)$. By Definition 5.17 and Lemma 5.3,

$$\sum_{T_i \in \tau} \mathsf{A}_\mathcal{J}(T_i, [t_0(\lambda), r_{\ell,q} + \Theta_\ell)) + \sum_{h=1}^{m} \mathsf{Res}([t_0(\lambda), r_{\ell,q} + \Theta_\ell))$$

$$\geq m \cdot (r_{\ell,q-\lambda+1} - t_0(\lambda)) + m \cdot |\Gamma_\lambda| + |\overline{\Gamma_\lambda}|$$

$$\{\text{by Lemma 5.4}\}$$

$$= m \cdot (r_{\ell,q-\lambda+1} - t_0(\lambda)) + m \cdot (r_{\ell,q} + \Theta_\ell - r_{\ell,q-\lambda+1} - \mathsf{W}_\mathcal{J}(T_\ell, r_{\ell,q-\lambda+1}) + 1 + \mu)$$

$$+ \mathsf{W}_\mathcal{J}(T_\ell, r_{\ell,q-\lambda+1}) - 1 - \mu$$

$$= m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - (m-1) \cdot (\mathsf{W}_\mathcal{J}(T_\ell, r_{\ell,q-\lambda+1}) - 1) + (m-1) \cdot \mu$$

$$\{\text{because } \mu \geq 0\}$$

$$\geq m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - (m-1) \cdot (\mathsf{W}_\mathcal{J}(T_\ell, r_{\ell,q-\lambda+1}) - 1). \tag{5.24}$$

Setting (5.22) into (5.24), we have

$$\sum_{T_i \in \tau} \mathsf{A}_\mathcal{J}(T_i, [t_0(\lambda), r_{\ell,q} + \Theta_\ell)) + m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - \mathcal{B}(r_{\ell,q} - t_0(\lambda) + \Theta_\ell)$$

$$\geq m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - (m-1) \cdot (\mathsf{W}_\mathcal{J}(T_\ell, r_{\ell,q-\lambda+1}) - 1).$$

Rearranging the terms in the above inequality, we have

$$\sum_{T_i \in \tau} \mathsf{A}_\mathcal{J}(T_i, [t_0(\lambda), r_{\ell,q} + \Theta_\ell)) + (m-1) \cdot (\mathsf{W}_\mathcal{J}(T_\ell, r_{\ell,q-\lambda+1}) - 1)$$

$$\geq \mathcal{B}(r_{\ell,q} - t_0(\lambda) + \Theta_\ell).$$

Setting $E_\ell^*(\lambda)$ and $M_\ell^*(r_{\ell,q} - t_0(\lambda))$ as defined in Definitions 5.20 and 5.21 into the inequality above, we have

$$M_\ell^*(r_{\ell,q} - t_0(\lambda)) + (m-1) \cdot (E_\ell^*(\lambda) - 1) \geq \mathcal{B}(r_{\ell,q} - t_0(\lambda) + \Theta_\ell).$$

Setting $r_{\ell,q} - t_0(\lambda) = \delta$ into the inequality above, we get (5.23).

Our remaining proof obligation is to establish the stated range for δ. Note that, by Definition 5.13, $\delta = r_{\ell,q} - t_0(\lambda) \geq r_{\ell,q} - r_{\ell,q-\lambda+1} \geq \mathcal{A}_\ell^{-1}(\lambda - 1)$, where the last inequality follows from Lemma 5.2. By

124

(5.20) and (5.23), we have for $k = \lambda$,

$$U_{sum} \cdot \delta + H_\ell + (m-1) \cdot (E_\ell^*(k) - 1) \geq \mathcal{B}(\delta + \Theta_\ell). \tag{5.25}$$

Applying (5.4) to (5.25), we have

$$U_{sum} \cdot \delta + H_\ell + (m-1) \cdot (E_\ell^*(k) - 1) \geq \max(0, \widehat{U} \cdot (\delta + \Theta_\ell - \sigma_{tot}))$$

$$\geq \widehat{U} \cdot (\delta + \Theta_\ell - \sigma_{tot}).$$

Solving the latter inequality for δ, we have $\delta \leq (H_\ell + (m-1) \cdot (E_\ell^*(k) - 1) + \widehat{U} \cdot \sigma_{tot} - \Theta_\ell \cdot \widehat{U}) / (\widehat{U} - U_{sum})$. Because δ is integral (as $r_{\ell,q}$ and $t_0(k)$ are integral), by Definition 5.23, $\delta \leq \delta_\ell^{\max}(k)$. The theorem follows. $\qquad\square$

Corollary 5.1. (Schedulability Test) *If, for each task $T_\ell \in \tau$, (5.23) does not hold for each $k \in [1, K_\ell]$ and $\delta \in [\mathcal{A}_\ell^{-1}(k-1), \delta_\ell^{\max}(k)]$, then no response-time bound is violated.*

Proof. The corollary follows from Theorem 5.3 and Definition 5.11, which implies $\lambda \in [1, K_\ell]$. $\qquad\square$

In Section 5.6, we improve the above schedulability test for fixed-job-priority preemptive schedulers such as GEDF and FIFO by replacing the term $(m-1) \cdot (E_\ell^*(k) - 1)$ in (5.23) with a smaller term proportional to $\max(m - F - 1, 0) \cdot E_\ell^*(k)$, where F is the number of processors that are always available (see Definition 5.6). This can be done because, under GEDF and FIFO, the problem job $T_{\ell,q}$ and its predecessors cannot be preempted by other jobs after a certain time point unless the competing demand carried from previous time instants is sufficiently large.

5.4.2   Step S3 (Calculating $M_\ell^*(\delta)$ and $E_\ell^*(k)$)

Note that we did not make any assumptions above about how jobs are scheduled except that the jobs of each task execute sequentially and jobs are prioritized as in Definition 5.7. Therefore, Corollary 5.1 is applicable to all fixed job-priority scheduling policies (these policies include preemptive variants of EDF, FIFO, static-priority policies, and their various combinations; non-preemptive variants can be supported similarly as discussed later in Section 5.4.3) provided the functions $M_\ell^*(\delta)$ (and its linear upper bound in Definition 5.22) and $E_\ell^*(k)$ are known. $M_\ell^*(\delta)$ and $E_\ell^*(k)$ can be derived for a particular algorithm by extending techniques from previously-published papers on the schedulability of sporadic tasks (Baruah, 2007; Leontyev and Anderson, 2008b) to incorporate more general arrival and execution patterns.

In this section, we derive the functions $E_\ell^*(k)$ and $M_\ell^*(\delta)$ for a fully preemptive prioritization scheme in which $\chi_{i,j} = r_{i,j} + D_i$, where D_i is a constant (preemptive global EDF and FIFO are the subcases of this scheme). Note that in this case the set $\mathcal{J}$ only contains jobs with higher or equal priority than that of $T_{\ell,q}$. We first prove some properties about jobs in the set $\mathcal{J}$.

Definition 5.24. Let $C_{i,k} = D_i - D_k$.

Lemma 5.5. *If $T_{\ell,q}$ violates its response-time bound and job $T_{a,b}$ is in $\mathcal{J}$, then $T_{a,b} \preceq T_{\ell,q}$ and $r_{a,b} \leq r_{\ell,q} + C_{\ell,a}$.*

Proof. Consider job $T_{a,b} \in \mathcal{J}$.

Case 1: $T_{a,b} \preceq T_{\ell,q}$.   By Definition 5.7, $r_{a,b} + D_a \leq r_{\ell,q} + D_\ell$. The required result follows from Definition 5.24.

Case 2: $T_{a,b} \succ T_{\ell,q}$.   By Definition 5.14, $T_{a,b}$ executes at some time $t \in [t_0(\lambda), r_{\ell,q} + \Theta_\ell)$ and IS_HP(t) holds. By (5.18), since $T_{a,b}$ executes at time t, there exists task $T_x \in \tau_p(t)$ such that job $T_{x,y}$ is ready at t, $T_{x,y} \preceq T_{\ell,q} \prec T_{a,b}$ and $T_{x,y}$ does not execute at t. This contradicts the assumption of full preemptivity. $\square$

Derivation of $M_\ell^*(\delta)$.   To derive $M_\ell^*(\delta)$, we first note that, by Lemma 5.5, only jobs $T_{a,b} \preceq T_{\ell,q}$ belong to $\mathcal{J}$ and can compete with $T_{\ell,q}$ or its predecessors.

Definition 5.25. Let T_{h,b_h} be the earliest pending job of T_h at time $t_0(k)$. We separate the tasks that may compete with $T_{\ell,q}$ into two disjoint sets:

$$\mathbf{HC} = \{T_h :: (T_{h,b_h} \text{ exists}) \wedge (r_{h,b_h} < t_0(k)) \wedge (T_{h,b_h} \in \mathcal{J})\};$$
$$\mathbf{NC} = \{T_h :: (r_{h,b_h} \geq t_0(k)) \wedge (T_{h,b_h} \in \mathcal{J})\}.$$

Here, **HC** denotes "high-priority carry-in" and **NC** denotes "non-carry-in".

Claim 5.7: $|\mathbf{HC}| \leq m - 1$.

Proof. By Definitions 5.12 and 5.25, $\mathbf{HC} \subseteq \tau_p(t_0(k) - 1)$. By Definition 5.13, all tasks in $\tau_p(t_0(k) - 1)$ execute at $t_0(k) - 1$ and $|\tau_p(t_0(k) - 1)| \leq m - 1$. Thus, $|\mathbf{HC}| \leq m - 1$. $\square$

Since the cumulative length of $[t_0(k), r_{\ell,q} + \Theta_\ell)$, depends on the difference $r_{\ell,q} - t_0(k)$, we use $\mathsf{A_{NC}}(T_i, r_{\ell,q} - t_0(k))$ and $\mathsf{A_{HC}}\ (T_i, r_{\ell,q} - t_0(k))$ to denote an upper-bound on $\mathsf{A}_{\mathcal{J}}(T_i, [t_0(k), r_{\ell,q} + \Theta_\ell))$ for the case when T_i is in $\mathbf{NC}$ and $\mathbf{HC}$, respectively. With this notation, we have

$$\sum_{T_i \in \tau} \mathsf{A}_{\mathcal{J}}(T_i, [t_0(\lambda), r_{\ell,q} + \Theta_\ell))$$

$$\leq \sum_{T_i \in \mathbf{HC}} \mathsf{A_{HC}}(T_i, r_{\ell,q} - t_0(\lambda)) + \sum_{T_i \in \mathbf{NC}} \mathsf{A_{NC}}(T_i, r_{\ell,q} - t_0(\lambda)). \tag{5.26}$$

We provide expressions for computing $\mathsf{A_{NC}}(T_i, \delta)$ and $\mathsf{A_{HC}}(T_i, \delta)$ in the following two lemmas. Their proofs can be found in the appendix.

Lemma 5.6: $\mathsf{A_{NC}}(T_i, \delta) = \min(\delta + \Theta_\ell, \gamma_i^u(\alpha_i^+(\delta + C_{\ell,i})))$.

Definition 5.26. Let $G_i(S, X) = \min(\gamma_i^u(S), \max(0, X - \mathcal{A}_\ell^{-1}(S-1)) + \gamma_i^u(S-1))$.

Lemma 5.7: $\mathsf{A_{HC}}(T_i, \delta) = \min(\delta + \Theta_\ell, G_i(\alpha_i^u(\delta + C_{\ell,i} + \Theta_i), \delta + C_{\ell,i} + \Theta_i))$.

To continue our derivation of $M_\ell^*(\delta)$, we set

$$M_\ell^*(\delta) = \max\left(\sum_{T_i \in \mathbf{HC}} \mathsf{A_{HC}}(T_i, \delta) + \sum_{T_i \in \mathbf{NC}} \mathsf{A_{NC}}(T_i, \delta)\right), \tag{5.27}$$

where max is taken over each choice of $\mathbf{HC}$ and $\mathbf{NC}$ subject to the following constraints.

$$\mathbf{NC} \cup \mathbf{HC} \subseteq \tau \wedge \mathbf{NC} \cap \mathbf{HC} = \emptyset \wedge |\mathbf{HC}| \leq m - 1 \tag{5.28}$$

The constraint $|\mathbf{HC}| \leq m - 1$ follows from Claim 5.7. It is easy to check that $0 \leq \mathsf{A_{NC}}(T_i, \delta)$ and $0 \leq \mathsf{A_{HC}}(T_i, \delta)$ for each $\delta \geq 0$. Thus, the sets maximizing the value $M_\ell^*(\delta)$ can be found by adding at most $m - 1$ tasks with the largest positive value of $\mathsf{A_{HC}}(T_i, \delta) - \mathsf{A_{NC}}(T_i, \delta)$ to $\mathbf{HC}$ and adding the remaining tasks to $\mathbf{NC}$.

By the selection of λ in Definition 5.11, (5.26), and (5.27), $M_\ell^*(r_{\ell,q} - t_0(\lambda))$ upper-bounds $\sum_{T_i \in \tau} \mathsf{A}_{\mathcal{J}}(T_i, [t_0(\lambda), r_{\ell,q} + \Theta_\ell))$ so it complies with Definition 5.21. In order to use Corollary 5.1, we are left with finding a constant H_ℓ such that (5.20) holds, so that $M_\ell^*(\delta)$ given by (5.27) complies with Definition 5.22.

Definition 5.27. Let $L_i(X) = \max(0, u_i \cdot X + \overline{e_i} \cdot B_i) + v_i$ for any X.

Lemma 5.8. (Proved in the appendix) *For all $\delta \geq 0$, $M_\ell^*(\delta) \leq U_{sum} \cdot \delta + H_\ell$, where $H_\ell = \sum_{T_i \in \tau} L_i(C_{\ell,i}) + U(m-1) \cdot \max(\Theta_i)$ and $U(y)$ is the sum of $\min(y, |\tau|)$ largest utilizations.*

We finally briefly discuss how $E_\ell^*(k)$ can be calculated.

Definition 5.28. Let $Q_\ell(k) = \max(0, \gamma_\ell^u(k-1) - 1) + \Theta_\ell$.

We set $E_\ell^*(k)$ as follows.

$$E_\ell^*(k) = G_\ell(\alpha_\ell^u(Q_\ell(k)), Q_\ell(k)) \tag{5.29}$$

In the lemma below, we show that $E_\ell^*(k)$ given by (5.29) complies with Definition 5.20.

Lemma 5.9. (Proved in the appendix) *If $E_\ell^*(k)$ is given by (5.29), then $E_\ell^*(\lambda) \geq \mathsf{W}_{\mathcal{J}}(T_i, r_{\ell, q-\lambda+1})$.*

Using an expression for H_ℓ given by Lemma 5.8, we can compute $\delta_\ell^{\max}(k)$ in Definition 5.23 for any given k. Given expressions for $\delta_\ell^{\max}(k)$, $M_\ell^*(\delta)$, and $E_\ell^*(k)$, we can apply Corollary 5.1 to check that each task $T_\ell \in \tau$ meets its response-time bound. In Section 5.5, we identify conditions under which the test is applicable and discuss its time complexity.

5.4.3 Analysis of Non-Preemptive Execution

As mentioned earlier, Corollary 5.1 is applicable if non-preemptive execution is allowed as well, provided the functions $M_\ell^*(\delta)$ (and its linear upper bound in Definition 5.22) and $E_\ell^*(k)$ are known. Additionally, all processors have to be fully available to tasks in τ because the semantics of non-preemptivity is not well-defined if a processor that executes a task in τ becomes unavailable. The derivation of $M_\ell^*(\delta)$ and $E_\ell^*(k)$ for the non-preemptive case would be similar to the procedures described above with the exception that $\mathcal{J}$ now may contain some jobs $T_{i,y} \succ T_{\ell,q}$.

5.5 Computational Complexity of the Test

According to Corollary 5.1, (5.23) needs to be checked for violation for all $k \in [1, K_\ell]$ and $\delta \in [\mathcal{A}_\ell^{-1}(k-1), \delta_\ell^{\max}(k)]$.

Theorem 5.4. *The time complexity of the presented test is pseudo-polynomial if there exists a constant c such that $U_{sum} \leq c < \widehat{U}$.*

Proof. We start with estimating the complexity of checking (5.23). The values of $\alpha_i^u(\Delta)$, $\gamma_i^u(k)$, $\mathcal{A}_i^{-1}(k)$, and $\mathcal{B}(\Delta)$ can be computed in constant time if $\alpha_i^u(\Delta)$, $\gamma_i^u(k)$, and $\mathcal{B}(\Delta)$ consist of an aperiodic and

128

periodic piecewise-linear parts. These assumptions are used in prior work on the Real-Time Calculus Toolbox (Wandeler and Thiele, 2006) and are sufficient for practical purposes. Under these assumptions, $M_\ell^*(\delta)$ for a given value of δ can be computed in $O(n)$ time, where n is the number of tasks, in two steps. First, for all tasks T_i, we calculate the values $\mathsf{A_{HC}}(T_i, \delta)$ and $\mathsf{A_{NC}}(T_i, \delta)$ in $O(n)$ time. Second, we calculate $\sum_{T_i \in \tau} \mathsf{A_{NC}}(T_i, \delta)$ in $O(n)$ time. Third, we select at most $m - 1$ largest positive values of $\mathsf{A_{HC}}(T_i, \delta) - \mathsf{A_{NC}}(T_i, \delta)$ in $O(n)$ time using linear-time selection (Blum et al., 1973) and add their sum to $\sum_{T_i \in \tau} \mathsf{A_{NC}}(T_i, \delta)$. The cost of checking (5.23) is thus $O(n)$.

For each task T_ℓ, the inequality (5.23) needs to be checked for all $k \in [1, K_\ell]$ and all integers in $[\mathcal{A}_\ell^{-1}(k-1), \delta_\ell^{\max}(k)]$. By Definition 5.23, $\delta_\ell^{\max}(k)$ is finite if its denominator is nonzero. By (5.5), we have $U_{sum} \leq \widehat{U}$. Therefore, $\delta_\ell^{\max}(k)$ is finite if (5.5) is strict. Overall, (5.23) has to be checked at most $n \cdot \max_{T_\ell \in \tau}(K_\ell \cdot \max_{k \leq K_\ell}(\delta_\ell^{\max}(k)))$ times, which implies the pseudo-polynomial time complexity. $\square$

Checking that (5.23) is violated for each integral value in $[\mathcal{A}_\ell^{-1}(k-1), \delta_\ell^{\max}(k)]$ can be computationally expensive. A fixed-point iterative technique can instead be applied so that only a (potentially small) subset of $[\mathcal{A}_\ell^{-1}(k-1), \delta_\ell^{\max}(k)]$ is checked. In essence, we skip intervals where (5.23) does not hold. A similar technique was used by Zhang and Burns (2009) for checking schedulability under uniprocessor EDF. The important difference is that our procedure does not rely on the assumptions of the sporadic task model and is applicable in multiprocessor systems.

In Definition 5.30 below, we define a sequence of values δ within the interval $[\mathcal{A}_\ell^{-1}(k-1), \delta_\ell^{\max}(k)]$ that need to be examined in order to check for a violation of (5.23) within this interval. We assume that $\mathcal{A}_\ell^{-1}(k-1) \leq \delta_\ell^{\max}(k)$, for otherwise (5.23) does not hold trivially. We will need an additional definition below.

Definition 5.29. Let $\mathcal{B}^{-1}(y) = \inf\{\Delta \mid \mathcal{B}(\Delta) > y\}$ be the pseudo-inverse function of the total processing capacity of the system.

Example 5.8. In Example 1.2, $\mathcal{B}^{-1}(2) = \inf\{\Delta \mid \mathcal{B}(\Delta) > 2\} = 5$.

Definition 5.30. Let $\xi(\delta) = \lfloor \mathcal{B}^{-1}(M_\ell^*(\delta) + (m-1) \cdot (E_\ell^*(k) - 1)) \rfloor - \Theta_\ell$. Let $\{x^{[n]}\}$ be the sequence such that $x^{[n+1]} := \xi(x^{[n]})$ and $x^{[1]} = \delta_\ell^{\max}(k)$.

Because, by Definition 5.20, $E_\ell^*(k)$ upper-bounds a positive variable (which includes the demand of the problem job $T_{\ell,q}$) and, by Definition 5.21, $M_\ell^*(\delta)$ upper-bounds a non-negative variable, $M_\ell^*(\delta) + (m-1) \cdot (E_\ell^*(k) - 1)$ is non-negative for each δ. Therefore, $\mathcal{B}^{-1}(M_\ell^*(\delta) + (m-1) \cdot (E_\ell^*(k) - 1))$ (and in turn $\xi(\delta)$) is well-defined for each δ. We henceforth assume that

(L) (5.23) does not hold for $\delta = \delta_\ell^{\max}(k) = x^{[1]}$.

Otherwise, the test in Corollary 5.1 fails trivially, when the first evaluation interval is considered.

Claim 5.8. $\xi(x)$ *is a non-decreasing function of* x.

Proof. The claim follows from the fact that $M_\ell^*(\delta)$ and $\mathcal{B}^{-1}(Y)$ are non-decreasing functions of their arguments. $\square$

Lemma 5.10: $\xi(x^{[1]}) \leq x^{[1]}$.

Proof. Consider $Z_1 = \mathcal{B}^{-1}(M_\ell^*(x^{[1]}) + (m-1) \cdot (E_\ell^*(k) - 1))$. By Definition 5.29,

$$
\begin{aligned}
Z_1 = {} & \inf\{\Delta \mid \mathcal{B}(\Delta) > M_\ell^*(x^{[1]}) + (m-1) \cdot (E_\ell^*(k) - 1)\} \\
& \{\text{from (L), we have } M_\ell^*(x^{[1]}) + (m-1) \cdot (E_\ell^*(k) - 1) < \mathcal{B}(x^{[1]} + \Theta_\ell)\} \\
\leq {} & x^{[1]} + \Theta_\ell.
\end{aligned}
$$

From the inequality above, we have

$$
\begin{aligned}
x^{[1]} \geq {} & Z_1 - \Theta_\ell \\
& \{\text{by the definition of } Z_1\} \\
= {} & \mathcal{B}^{-1}(M_\ell^*(x^{[1]}) + (m-1) \cdot (E_\ell^*(k) - 1)) - \Theta_\ell \\
\geq {} & \left\lfloor \mathcal{B}^{-1}(M_\ell^*(x^{[1]}) + (m-1) \cdot (E_\ell^*(k) - 1)) \right\rfloor - \Theta_\ell \\
= {} & \xi(x^{[1]}). \hspace{4cm} \square
\end{aligned}
$$

Lemma 5.11. $x^{[n+1]} \leq x^{[n]}$ *for each* n.

Proof. **Base case:** $n = 1$. Because, by Definition 5.30, $x^{[2]} = \xi(x^{[1]})$, the required result immediately follows from Lemma 5.10.

Induction step: $n > 1$. By the induction hypothesis, $x^{[n]} \leq x^{[n-1]}$. By Claim 5.8, we have $\xi(x^{[n]}) \leq \xi(x^{[n-1]})$. By Definition 5.30, this implies $x^{[n+1]} \leq x^{[n]}$. $\square$

We next prove an auxiliary lemma.

Lemma 5.12. *If* $y > \mathcal{B}^{-1}(y_0)$, *then* $\mathcal{B}(y) > y_0$.

Proof. Let $y^* = \inf\{\Delta \mid \mathcal{B}(\Delta) > y_0\}$. This implies that

$$\mathcal{B}(y) \le y_0 \text{ for each } y < y^*. \tag{5.30}$$

We now consider two cases.

Case 1: $\mathcal{B}(y^*) > y_0$. Because $\mathcal{B}(\Delta)$ is non-decreasing, by the condition of the case, for $y > y^*$, $\mathcal{B}(y) \ge \mathcal{B}(y^*) > y_0$.

Case 2: $\mathcal{B}(y^*) = y_0$. Suppose, contrary to the statement of the lemma, that there exists $y' > y^*$ such that $\mathcal{B}(y') \le y_0$. Then, because $\mathcal{B}(\Delta)$ is non-decreasing, $\mathcal{B}(\Delta) \le y_0$ for each $\Delta \in [y^*, y']$, and hence, by (5.30), $\mathcal{B}(\Delta) \le y_0$ for each $\Delta \le y'$. Therefore, y^* is not an infimum for the set where $\mathcal{B}(\Delta) > y_0$, which contradicts the definition of y^*. $\qquad\square$

Lemma 5.13. *If $x^{[n+1]} < x^{[n]}$, then (5.23) does not hold for each non-negative integral $\delta \in (x^{[n+1]}, x^{[n]}]$.*

Proof. Consider a non-negative $\delta \in (x^{[n+1]}, x^{[n]}]$. We first lower-bound $\delta + \Theta_\ell$ as follows.

$$\delta + \Theta_\ell > x^{[n+1]} + \Theta_\ell$$

$$\{\text{because } x^{[n+1]} = \xi(x^n), \text{ by Definition 5.30}\}$$

$$= \left\lfloor \mathcal{B}^{-1}(M_\ell^*(x^{[n]}) + (m-1) \cdot (E_\ell^*(k) - 1)) \right\rfloor - \Theta_\ell + \Theta_\ell$$

$$= \left\lfloor \mathcal{B}^{-1}(M_\ell^*(x^{[n]}) + (m-1) \cdot (E_\ell^*(k) - 1)) \right\rfloor$$

Because δ and Θ_ℓ are integral, $\delta + \Theta_\ell > \mathcal{B}^{-1}(M_\ell^*(x^{[n]}) + (m-1) \cdot (E_\ell^*(k) - 1))$. By Lemma 5.12, the last inequality implies

$$\mathcal{B}(\delta + \Theta_\ell) > M_\ell^*(x^{[n]}) + (m-1) \cdot (E_\ell^*(k) - 1)$$

$$\{\text{by the selection of } \delta \text{ and } M^* \text{ being non-decreasing}\}$$

$$\ge M_\ell^*(\delta) + (m-1) \cdot (E_\ell^*(k) - 1). \qquad\square$$

The following theorem gives a method for checking (5.23) on the interval $[\mathcal{A}_\ell^{-1}(k-1), \delta_\ell^{\max}(k)]$ which skips sub-intervals where (5.23) does not hold.
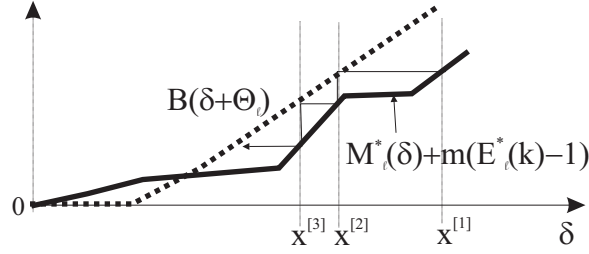
Figure 5.4: Iterative process for finding δ_ℓ in Example 5.9.

Theorem 5.5. *Let $\{x^{[n]}\}$ be the sequence defined in Definition 5.30. If $x^{[n+1]} < \mathcal{A}_\ell^{-1}(k-1)$, then (5.23) does not hold for each integral δ within the interval $[\mathcal{A}_\ell^{-1}(k-1), \delta_\ell^{\max}(k)]$.*

Proof. The theorem follows from dividing the interval $(x^{[n+1]}, \delta_\ell^{\max}(k)]$ into subintervals $(x^{[i+1]}, x^{[i]}]$ and applying Lemma 5.13 to each of the subintervals. □

We proved the above theorem for the case when time is integral. We defer consideration of continuous time to future work. According to Theorem 5.5, we can apply Corollary 5.1 as follows. First, we check whether (5.23) does not hold for $\delta = \delta_\ell^{\max}(k)$. Second, we construct the sequence $\{x^{[n]}\}$ as defined in Definition 5.30. If a fixed point $x^{[n]} = x^{[n+1]}$ is not found in the interval $[\mathcal{A}_\ell^{-1}(k-1), \delta_\ell^{\max}(k)]$, then, by Theorem 5.5, (5.23) does not hold for each $\delta \in [\mathcal{A}_\ell^{-1}(k-1), \delta_\ell^{\max}(k)]$. If such a fixed point is found, then we conservatively claim that the response-time bound Θ_ℓ is violated.

Example 5.9. The iteration process described above can be illustrated graphically. Figure 5.4 shows two functions of δ: $\mathcal{B}(\delta + \Theta_\ell)$ and $M_\ell^*(\delta) + (m-1) \cdot (E_\ell^*(k) - 1)$, which are depicted with bold dotted and solid lines, respectively. The iteration process starts with $x^{[1]} = \delta_\ell^{\max}(k)$. At this point, $M_\ell^*(x^{[1]}) + (m-1) \cdot (E_\ell^*(k) - 1) < \mathcal{B}(x^{[1]} + \Theta_\ell)$. The next step is to set $x^{[2]} = \xi(x^{[1]})$ as shown. Similarly, $x^{[3]}$ is computed. The process continues until a fixed point is found or $x^{[n+1]} < \mathcal{A}_\ell^{-1}(k-1)$ holds. Thus, the iterations skip portions of the interval $[\mathcal{A}_\ell^{-1}(k-1), \delta_\ell^{\max}(k)]$ where (5.23) is guaranteed to fail.

5.6 Schedulability Test for GEDF-like Schedulers

In this section, we improve Inequality (5.23) for a prioritization scheme in which $\chi_{i,j} = r_{i,j} + D_i$, where D_i is a constant. We do this by more carefully estimating $\mathcal{J}$-allocations within the intervals $[t_0(\lambda), r_{\ell, q-\lambda+1}) \cup \Gamma_\lambda$ and $\overline{\Gamma_\lambda}$. We divide these intervals into four non-intersecting sets and estimate the $\mathcal{J}$-allocations individually within these sets in Lemmas 5.14–5.17. Using the obtained results, we establish Theorem 5.6, which gives a necessary condition for a response-time bound violation. This

theorem is proved similarly to Theorem 5.3. Finally, Corollary 5.3 gives us an improved schedulability test for GEDF-like schedulers.

Definition 5.31. Let $C_\ell = \max_{T_i \in \tau}(D_\ell - D_i)$.

In Definition 5.32 and Lemmas 5.14–5.17 below, we assume that $\Theta_\ell > \gamma_\ell^u(\lambda) + C_\ell$ holds. In this case, we can improve Inequality (5.23) by replacing the term $(m-1) \cdot E_\ell^*(k)$ with a smaller term proportional to $\max(m - F - 1, 0) \cdot E_\ell^*(k)$, where F is the number of fully available processors. (If $\Theta_\ell \leq \gamma_\ell^u(\lambda) + C_\ell$, then Theorem 5.3 can be applied to check for a response-time bound violation.)

Definition 5.32. Let $\Gamma_\lambda^{[1]} = [r_{\ell,q-\lambda+1}, r_{\ell,q} + C_\ell) \cap \Gamma_\lambda$, $\Gamma_\lambda^{[2]} = [r_{\ell,q} + C_\ell, r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda)) \cap \Gamma_\lambda$, and $\Gamma_\lambda^{[3]} = [r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda), r_{\ell,q} + \Theta_\ell) \cap \Gamma_\lambda$, as shown in Figure 5.5.

Additionally, let $\overline{\Gamma_\lambda}^{[1]} = [r_{\ell,q-\lambda+1}, r_{\ell,q} + C_\ell) \cap \overline{\Gamma_\lambda}$, $\overline{\Gamma_\lambda}^{[2]} = [r_{\ell,q} + C_\ell, r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda)) \cap \overline{\Gamma_\lambda}$, and $\overline{\Gamma_\lambda}^{[3]} = [r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda), r_{\ell,q} + \Theta_\ell) \cap \overline{\Gamma_\lambda}$.

Note that, by Definition 5.32,

$$[t_0(\lambda), r_{\ell,q} + \Theta_\ell) = [t_0(\lambda), r_{\ell,q-\lambda+1}) \cup \Gamma_\lambda \cup \overline{\Gamma_\lambda}^{[1]} \cup \overline{\Gamma_\lambda}^{[2]} \cup \overline{\Gamma_\lambda}^{[3]}. \tag{5.31}$$

In the rest of this section we let μ be defined as in Lemma 5.4.

Lemma 5.14: $\widehat{\mathsf{A}_\mathcal{J}}([t_0(\lambda), r_{\ell,q-\lambda+1}) \cup \Gamma_\lambda) = m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - m \cdot (\mathsf{W}_\mathcal{J}\,(T_\ell, r_{\ell,q-\lambda+1}) - 1) + m \cdot \mu$.

Proof. By Definition 5.17, we have

$$\widehat{\mathsf{A}_\mathcal{J}}([t_0(\lambda), r_{\ell,q-\lambda+1}) \cup \Gamma_\lambda)$$
$$= \widehat{\mathsf{A}_\mathcal{J}}([t_0(\lambda), r_{\ell,q-\lambda+1})) + \widehat{\mathsf{A}_\mathcal{J}}(\Gamma_\lambda)$$
$$\{\text{by Definition 5.18 and Claim 5.4}\}$$
$$= m \cdot (r_{\ell,q-\lambda+1} - t_0(\lambda)) + m \cdot |\Gamma_\lambda|$$
$$\{\text{by Lemma 5.4}\}$$
$$= m \cdot (r_{\ell,q-\lambda+1} - t_0(\lambda)) + m \cdot (r_{\ell,q} + \Theta_\ell - r_{\ell,q-\lambda+1} - \mathsf{W}_\mathcal{J}(T_\ell, r_{\ell,q-\lambda+1}) + 1 + \mu)$$
$$= m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - m \cdot (\mathsf{W}_\mathcal{J}(T_\ell, r_{\ell,q-\lambda+1}) - 1) + m \cdot \mu. \qquad \square$$

Lemma 5.15: $\widehat{\mathsf{A}_\mathcal{J}}(\overline{\Gamma_\lambda}^{[1]}) \geq r_{\ell,q} + C_\ell - r_{\ell,q-\lambda+1} - |\Gamma_\lambda^{[1]}|$.

Proof. By Definitions 5.18 and 5.32, $\overline{\Gamma_\lambda}^{[1]} = [r_{\ell,q-\lambda+1}, r_{\ell,q} + C_\ell) \cap \overline{\Gamma_\lambda} = [r_{\ell,q-\lambda+1}, r_{\ell,q} + C_\ell) \setminus \Gamma_\lambda = [r_{\ell,q-\lambda+1}, r_{\ell,q} + C_\ell) \setminus \Gamma_\lambda^{[1]}$. By Claim 5.5, T_ℓ executes at each instant within $\overline{\Gamma_\lambda}$, and hence, at each instant
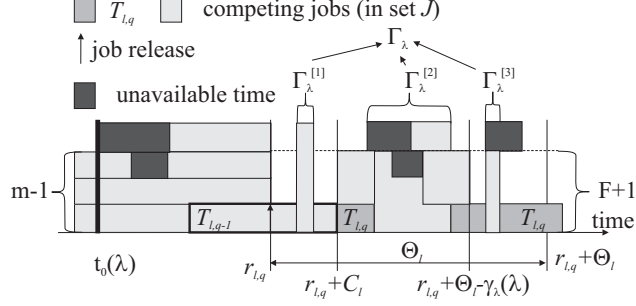
133

Figure 5.5: Conditions for a response-time bound violation for $\lambda = 1$.

within $[r_{\ell,q-\lambda+1}, r_{\ell,q} + C_\ell) \setminus \Gamma_\lambda^{[1]}$. Thus, $\mathsf{A}_{\mathcal{J}}(T_\ell, [r_{\ell,q-\lambda+1}, r_{\ell,q} + C_\ell) \setminus \Gamma_\lambda^{[1]}) = r_{\ell,q} + C_\ell - r_{\ell,q-\lambda+1} - |\Gamma_\lambda^{[1]}|$. The required result follows from Definition 5.17. $\qquad\square$

Lemma 5.16: $\widehat{\mathsf{A}_{\mathcal{J}}}(\overline{\Gamma_\lambda}^{[3]}) \geq \gamma_\ell^u(\lambda) - |\Gamma_\lambda^{[3]}|$.

Proof. By Definitions 5.18 and 5.32, $\overline{\Gamma_\lambda}^{[3]} = [r_{\ell,q}+\Theta_\ell-\gamma_\ell^u(\lambda), r_{\ell,q}+\Theta_\ell) \cap \overline{\Gamma_\lambda} = [r_{\ell,q}+\Theta_\ell-\gamma_\ell^u(\lambda), r_{\ell,q}+\Theta_\ell)\setminus \Gamma_\lambda = [r_{\ell,q}+\Theta_\ell-\gamma_\ell^u(\lambda), r_{\ell,q}+\Theta_\ell)\setminus\Gamma_\lambda^{[3]}$. By Claim 5.5, T_ℓ executes at each instant within $\overline{\Gamma_\lambda}$, and hence, at each instant within $[r_{\ell,q}+\Theta_\ell-\gamma_\ell^u(\lambda), r_{\ell,q}+\Theta_\ell)\setminus\Gamma_\lambda^{[3]}$. Thus, $\mathsf{A}_{\mathcal{J}}(T_\ell, [r_{\ell,q}+\Theta_\ell-\gamma_\ell^u(\lambda), r_{\ell,q}+\Theta_\ell)\setminus\Gamma_\lambda^{[3]}) = \gamma_\ell^u(\lambda) - |\Gamma_\lambda^{[3]}|$. The required result follows from Definition 5.17. $\qquad\square$

If $\Gamma_\lambda^{[3]} = \emptyset$, then, because by Definition 5.11, $f_{\ell,q-\lambda} \leq r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda)$, jobs $T_{\ell,q-\lambda+1}, \ldots, T_{\ell,q}$ can execute uninterruptedly within $[r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda), r_{\ell,q} + \Theta_\ell)$. As their total execution time is at most $\gamma_\ell^u(\lambda)$, $T_{\ell,q}$ will finish by $r_{\ell,q} + \Theta_\ell$ leading to a contradiction. We henceforth assume $|\Gamma_\lambda^{[3]}| > 0$.

From Lemma 5.5, the corollary below follows.

Corollary 5.2. *No job in $\mathcal{J}$ is released after $r_{\ell,q} + \max_{T_i \in \tau}(D_\ell - D_i)$.*

Proof. Consider job $T_{i,j} \in \mathcal{J}$. By Lemma 5.5, $r_{i,j} \leq r_{\ell,q} + C_{\ell,i}$. Thus, $r_{i,j} \leq r_{\ell,q} + D_\ell - D_i \leq r_{\ell,q} + \max_{T_i \in \tau}(D_\ell - D_i)$. $\qquad\square$

Definition 5.33. Let $a = \min(F + 1, m)$. (Recall that F is the number of fully available processors as defined in Definition 5.6.)

Lemma 5.17: $\widehat{\mathsf{A}_{\mathcal{J}}}(\overline{\Gamma_\lambda}^{[2]}) \geq a \cdot (-C_\ell - \gamma_\ell^u(\lambda) - r_{\ell,q} + r_{\ell,q-\lambda+1} + \mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1 - \mu) + a \cdot (|\Gamma_\lambda^{[1]}| + |\Gamma_\lambda^{[3]}|)$.

Proof. We first note that, by Definitions 5.18 and 5.32, we have

$$\overline{\Gamma_\lambda}^{[2]} = [r_{\ell,q} + C_\ell, r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda)) \setminus \Gamma_\lambda. \tag{5.32}$$

134

By Corollary 5.2 and Definition 5.31, nor job in $\mathcal{J}$ nor its predecessors can be released after $r_{\ell,q} + C_\ell$. If at most F available processors execute jobs in $\mathcal{J}$ at some time instant $t' \in [r_{\ell,q} + C_\ell, r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda)) \setminus \Gamma_\lambda$, then at each time $t \geq t'$ all tasks in $\tau_p(t)$ with ready jobs in $\mathcal{J}$ can be accommodated using F fully available processors. By Claim 5.3, this implies that jobs of T_ℓ execute uninterruptedly within $[t', r_{\ell,q} + \Theta_\ell)$. The completion time of $T_{\ell,q}$ is thus

$$
\begin{aligned}
f_{\ell,q} &\leq \max(t', f_{\ell,q-\lambda}) + \gamma_\ell^u(\lambda) \\
&\qquad \text{\{by Definition 5.11\}} \\
&\leq \max(t', r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda)) + \gamma_\ell^u(\lambda) \\
&\qquad \text{\{by the selection of } t'\} \\
&\leq r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda) + \gamma_\ell^u(\lambda) \\
&= r_{\ell,q} + \Theta_\ell,
\end{aligned}
$$

leading to a contradiction.

We henceforth assume that at least $a = \min(F+1, m)$ available processors execute jobs in $\mathcal{J}$ at each time within $[r_{\ell,q} + C_\ell, r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda)) \setminus \Gamma_\lambda$ (see Figure 5.5). Thus,

$$
\begin{aligned}
\widehat{\mathsf{A}_{\mathcal{J}}}(\overline{\Gamma_\lambda}^{[2]}) &\geq a \cdot |\overline{\Gamma_\lambda}^{[2]}| \\
&\qquad \text{\{by (5.32)\}} \\
&= a \cdot |[r_{\ell,q} + C_\ell, r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda)) \setminus \Gamma_\lambda| \\
&= a \cdot (\Theta_\ell - \gamma_\ell^u(\lambda) - C_\ell - (|\Gamma_\lambda| - |\Gamma_\lambda^{[1]}| - |\Gamma_\lambda^{[3]}|)) \\
&= a \cdot (\Theta_\ell - \gamma_\ell^u(\lambda) - C_\ell - |\Gamma_\lambda|) + a \cdot (|\Gamma_\lambda^{[1]}| + |\Gamma_\lambda^{[3]}|) \\
&\qquad \text{\{by Lemma 5.4\}} \\
&= a \cdot (\Theta_\ell - \gamma_\ell^u(\lambda) - C_\ell - (r_{\ell,q} + \Theta_\ell - r_{\ell,q-\lambda+1} \\
&\qquad - \mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) + 1 + \mu)) + a \cdot (|\Gamma_\lambda^{[1]}| + |\Gamma_\lambda^{[3]}|) \\
&= a \cdot (-\gamma_\ell^u(\lambda) - C_\ell - r_{\ell,q} + r_{\ell,q-\lambda+1} \\
&\qquad + \mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1 - \mu) + a \cdot (|\Gamma_\lambda^{[1]}| + |\Gamma_\lambda^{[3]}|). \qquad \square
\end{aligned}
$$

Claim 5.9: $r_{\ell,q} - r_{\ell,q-\lambda+1} \leq \max(0, \gamma_\ell^u(\lambda - 1) - 1)$. (Note that this result does not depend on the scheduler being assumed.)

Proof. If $\lambda = 1$, then $r_{\ell,q} - r_{\ell,q-\lambda+1} = 0$. Alternatively, if $\lambda > 1$, then, by (5.12), $r_{\ell,q-\lambda+1} + \Theta_\ell \geq f_{\ell,q-\lambda+1} > r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda - 1)$, where the last inequality follows from Definition 5.11. Therefore, $r_{\ell,q} - r_{\ell,q-\lambda+1} \leq \gamma_\ell^u(\lambda - 1) - 1$, as time is integral. □

The following definition is used to define the schedulability test for GEDF-like schedulers in Theorem 5.6 and Corollary 5.3 below.

Definition 5.34. Let

$$
Z_h(k) = \begin{cases}
(m - 1) \cdot (E_h^*(k) - 1) & \text{if } \Theta_h \leq \gamma_h^u(k) + C_\ell, \\
\min \big((m - 1) \cdot (E_h^*(k) - 1), & \\
(m - a) \cdot (E_h^*(k) - 1) + (a - 1) \cdot (\gamma_h^u(k) + \max(0, \gamma_h^u(k-1) - 1) + C_h) \big) & \\
& \text{otherwise,}
\end{cases}
$$

where a is defined as in Definition 5.33.

Theorem 5.6. *If the response-time bound Θ_ℓ of $T_{\ell,q}$ is violated (as we have assumed), then for some $k = \lambda$ and δ such that $\delta \geq \mathcal{A}_\ell^{-1}(k - 1)$ and $\delta \leq \lfloor (H_\ell + Z_\ell(k) + \widehat{U} \cdot \sigma_{tot} - \Theta_\ell \cdot \widehat{U})/(\widehat{U} - U_{sum}) \rfloor$, (5.33) below holds.*

$$
M_\ell^*(\delta) + Z_\ell(k) \geq \mathcal{B}(\delta + \Theta_\ell), \tag{5.33}
$$

Proof. Consider job $T_{\ell,q}$, $k = \lambda$, and time instants $r_{\ell,q-\lambda+1}$ and $t_0(\lambda)$ as defined in Definitions 5.11 and 5.13. We let $\delta = r_{\ell,q} - t_0(\lambda)$. We consider two cases.

Case 1: $\Theta_\ell \leq \gamma_\ell^u(\lambda) + \Theta_\ell$. By Theorem 5.3, (5.34) below holds

$$
M_\ell^*(\delta) + (m - 1) \cdot (E_\ell^*(\lambda) - 1) \geq \mathcal{B}(\delta + \Theta_\ell). \tag{5.34}
$$

Case 2: $\Theta_\ell > \gamma_\ell^u(\lambda) + \Theta_\ell$. By Definition 5.17, we have,

$$
\sum_{T_i \in \tau} \mathsf{A}_{\mathcal{J}}(T_i, [t_0(\lambda), r_{\ell,q} + \Theta_\ell)) + \sum_{h=1}^{m} \mathsf{Res}([t_0(\lambda), r_{\ell,q} + \Theta_\ell))
$$
$$
= \widehat{\mathsf{A}_{\mathcal{J}}}(T_i, [t_0(\lambda), r_{\ell,q} + \Theta_\ell))
$$
$$
\{\text{by (5.31)}\}
$$

136

$$= \widehat{\mathsf{A}_{\mathcal{J}}}([t_0(\lambda), r_{\ell,q-\lambda+1}) \cup \Gamma_\lambda) + \widehat{\mathsf{A}_{\mathcal{J}}}(\overline{\Gamma_\lambda}^{[1]}) + \widehat{\mathsf{A}_{\mathcal{J}}}(\overline{\Gamma_\lambda}^{[2]}) + \widehat{\mathsf{A}_{\mathcal{J}}}(\overline{\Gamma_\lambda}^{[3]})$$

{by Lemmas 5.14–5.17}

$$\geq m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - m \cdot (\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1) + m \cdot \mu$$

$$+ r_{\ell,q} + C_\ell - r_{\ell,q-\lambda+1} - |\Gamma_\lambda^{[1]}|$$

$$+ a \cdot (-C_\ell - \gamma_\ell^u(\lambda) - r_{\ell,q} + r_{\ell,q-\lambda+1} + \mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1 - \mu)$$

$$+ a \cdot (|\Gamma_\lambda^{[1]}| + |\Gamma_\lambda^{[3]}|) + \gamma_\ell^u(\lambda) - |\Gamma_\lambda^{[3]}|$$

$$= m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - (m - a) \cdot (\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1)$$

$$+ (m - a) \cdot \mu + (a - 1) \cdot (|\Gamma_\lambda^{[1]}| + |\Gamma_\lambda^{[3]}|)$$

$$+ (1 - a) \cdot (\gamma_\ell^u(\lambda) + C_\ell + r_{\ell,q} - r_{\ell,q-\lambda+1})$$

{because $\mu \geq 0$ and $|\Gamma_\lambda^{[1]}| + |\Gamma_\lambda^{[3]}| \geq 0$}

$$\geq m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - (m - a) \cdot (\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1)$$

$$+ (1 - a) \cdot (\gamma_\ell^u(\lambda) + C_\ell + r_{\ell,q} - r_{\ell,q-\lambda+1}). \tag{5.35}$$

Setting (5.22) into (5.35), we have

$$\sum_{T_i \in \tau} \mathsf{A}_{\mathcal{J}}(T_i, [t_0(\lambda), r_{\ell,q} + \Theta_\ell))$$

$$+ m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - \mathcal{B}(r_{\ell,q} - t_0(\lambda) + \Theta_\ell)$$

$$\geq m \cdot (r_{\ell,q} - t_0(\lambda) + \Theta_\ell) - (m - a) \cdot (\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1)$$

$$+ (1 - a) \cdot (\gamma_\ell^u(\lambda) + C_\ell + r_{\ell,q} - r_{\ell,q-\lambda+1}).$$

Rearranging the terms in the above inequality, we have

$$\sum_{T_i \in \tau} \mathsf{A}_{\mathcal{J}}(T_i, [t_0(\lambda), r_{\ell,q} + \Theta_\ell)) + (m - a) \cdot (\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1)$$

$$+ (a - 1) \cdot (\gamma_\ell^u(\lambda) + C_\ell + r_{\ell,q} - r_{\ell,q-\lambda+1})$$

$$\geq \mathcal{B}(r_{\ell,q} - t_0(\lambda) + \Theta_\ell).$$

From Claim 5.9, we therefore have

$$\sum_{T_i \in \tau} \mathsf{A}_{\mathcal{J}}(T_i, [t_0(\lambda), r_{\ell,q} + \Theta_\ell)) + (m - a) \cdot (\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) - 1)$$

$$+ (a-1) \cdot (\gamma_\ell^u(\lambda) + C_\ell + \max(0, \gamma_\ell^u(\lambda-1) - 1))$$

$$\geq \mathcal{B}(r_{\ell,q} - t_0(\lambda) + \Theta_\ell).$$

Setting $E_\ell^*(\lambda)$ and $M_\ell^*(r_{\ell,q} - t_0(\lambda))$ as defined in Definitions 5.20 and 5.21 into the inequality above, we get

$$M_\ell^*(r_{\ell,q} - t_0(\lambda))$$
$$+ (m-a) \cdot (E_\ell^*(\lambda) - 1) + (a-1) \cdot (\gamma_\ell^u(\lambda) + C_\ell + \max(0, \gamma_\ell^u(\lambda-1) - 1))$$
$$\geq \mathcal{B}(r_{\ell,q} - t_0(\lambda) + \Theta_\ell).$$

Setting $\delta = r_{\ell,q} - t_0(\lambda)$ in the inequality above, we have

$$M_\ell^*(\delta) + (m-a) \cdot (E_\ell^*(\lambda) - 1) + (a-1) \cdot (\gamma_\ell^u(\lambda) + C_\ell + \max(0, \gamma_\ell^u(\lambda-1) - 1))$$
$$\geq \mathcal{B}(\delta + \Theta_\ell). \tag{5.36}$$

Additionally, (5.34) holds by Theorem 5.3. Combining (5.34) and (5.36) using Definition 5.34, we get (5.33). The stated range for δ can further be found similarly to Theorem 5.3. $\qquad\square$

From Theorem 5.6, an improved schedulability test follows.

Corollary 5.3. (Improved Schedulability Test) *Let* $\delta_h^{\max}(k)' = \left\lfloor (H_h + Z_h(k) + \widehat{U} \cdot \sigma_{tot} - \Theta_h \cdot \widehat{U}) / (\widehat{U} - U_{sum}) \right\rfloor$. *If, for each task* $T_h \in \tau$, $M_h^*(\delta) + Z_h(k) < \mathcal{B}(\delta + \Theta_h)$ *for each* $k \in [1, K_h]$ *and* $\delta \in [\mathcal{A}_h^{-1}(k-1), \delta_h^{\max}(k)']$, *then no response-time bound is violated.*

By Definition 5.34, for large values of the response-time bound Θ_h such that $\Theta_h > \gamma_h^u(k) + C_h$, $Z_h(k)$ is $\min\left((m-1) \cdot (E_h^*(k) - 1), (m-a) \cdot (E_h^*(k) - 1) + (a-1) \cdot (\gamma_h^u(k) + \max(0, \gamma_h^u(k-1) - 1) + C_h)\right)$. This value is smaller than $(m-1) \cdot (E_h^*(k) - 1)$ for large values of Θ_h because $E_h^*(k)$ is proportional to Θ_h by Lemma 5.9. Thus, the schedulability test given in Corollary 5.3 is less pessimistic for large response-time bounds than the test in Corollary 5.1. In the next section, we use the improved schedulability test to derive closed-form expressions for response-time bounds.

5.7 Closed-Form Expressions for Response-Time Bounds

Though the iterative procedure described in Section 5.5 can significantly reduce the time needed to check response-time bounds using Corollaries 5.1 and 5.3, the verification time can still be large if the task set is large and tasks have complex job arrival and execution-time patterns. In this section, we further reduce the computation time by deriving closed-form expressions for the response-time bounds Θ_i under GEDF-like schedulers. In Chapter 3, it has been shown that GEDF (and many other schedulers) ensures a maximum response-time bound of $x + p_i + e_i^{max}$, where $x \geq 0$, for each sporadic task $T_i \in \tau$, if tasks have implicit deadlines, all processors are fully available, and $U_{sum} \leq m$. In this chapter, we prove a similar result for systems specified as in Section 5.1. We will be seeking response-time bounds of the form $\Theta_i = x + \gamma_i^u(K_i) + C_i$, where $x > 0$, and K_i and C_i are as defined in Definitions 5.3 and 5.31. In the rest of this section, we derive x based upon the task parameters and resource availability. The derivation process is similar to finding an upper bound on δ in Theorem 5.3. In Lemmas 5.18 and 5.19 below, we first establish upper bounds on $E_\ell^*(k)$ and $M_\ell^*(\delta)$ as functions of x for the case when the response-time bound is a function of x. We then set the obtained expressions into the schedulability test and solve the resulting inequality for x.

Definition 5.35. Let $Y_\ell = L_\ell(\max(0, \gamma_\ell^u(K_\ell - 1) - 1) + \gamma_\ell^u(K_\ell) + C_\ell)$, where L is defined as in Definition 5.27.

Lemma 5.18. (Proved in the appendix) *If* $\Theta_\ell = x + \gamma_\ell^u(K_\ell) + C_\ell$, *then* $E_\ell^*(k) \leq Y_\ell + u_\ell \cdot x$ *for* $k \in [1, K_\ell]$.

Definition 5.36. Let $\mathcal{W}$ be the sum of $m - 1$ largest values $u_i \cdot (\gamma_i^u(K_i) + C_i)$.

Lemma 5.19. (Proved in the appendix) *If* $\Theta_i = x + \gamma_i^u(K_i) + C_i$ *for each task* T_i *and* $\delta \geq 0$, *then* $M_\ell^*(\delta) \leq U_{sum} \cdot \delta + U(m-1) \cdot x + \mathcal{W} + \sum_{T_i \in \tau} L_i(C_{\ell,i})$, *where* $U(m-1)$ *is the sum of* $m - 1$ *largest task utilizations.*

Theorem 5.7. *If* $\widehat{U} - (m - a) \cdot \max(u_i) - U(m-1) > 0$ *and* $U_{sum} \leq \widehat{U}$, *then, under a GEDF-like scheduler, the maximum response time of any job of* T_i *is at most* $x + \gamma_i^u(K_i) + C_i$, *where*

$$x = \max_{T_h \in \tau} \left(\frac{\mathcal{W} + \widehat{U} \cdot \sigma_{tot} + V_h + \sum_{T_i \in \tau} L_i(C_{h,i})}{\widehat{U} - (m-a) \cdot u_h - U(m-1)} \right) + 1 \tag{5.37}$$

and $V_h = (m - a) \cdot (Y_h - 1) + (a - 1 - \widehat{U}) \cdot (\gamma_h^u(K_h) + C_h) + (a - 1) \cdot \max(0, \gamma_h^u(K_h - 1) - 1)$.

Proof. Suppose to the contrary that task T_ℓ violates its response-time bound $\Theta_\ell = x + \gamma_\ell^u(K_\ell) + C_\ell$. Because $x > 0$, and $\gamma_\ell^u(K_\ell) \geq \gamma_\ell^u(k)$ for each $k \in [1, K_\ell]$, we have

$$\Theta_\ell > \gamma_\ell^u(k) + C_\ell \text{ for each } k \in [1, K_\ell]. \tag{5.38}$$

By Theorem 5.6, for some $k \in [1, K_\ell]$ (particularly, for $k = \lambda$ as defined in Definition 5.11) and $\delta \geq 0$, (5.33) holds. (Note that $\delta \geq \mathcal{A}_\ell^{-1}(k-1)$ by Theorem 5.6 and $\mathcal{A}_\ell^{-1}(k-1) \geq 0$ by Definition 5.3.) Setting $k = \lambda$ and the bound for $\mathcal{B}$ given by (5.4) into (5.33), we have

$$M_\ell^*(\delta) + Z_\ell(\lambda) \geq \widehat{U} \cdot (\delta + \Theta_\ell - \sigma_{tot}).$$

Because $\Theta_\ell > \gamma_\ell^u(\lambda) + C_\ell$ by (5.38), from Definition 5.34 and the inequality above, we have

$$M_\ell^*(\delta) + (m-a) \cdot (E_\ell^*(\lambda) - 1) + (a-1) \cdot (\gamma_\ell^u(\lambda) + C_\ell + \max(0, \gamma_\ell^u(\lambda-1) - 1))$$
$$\geq \widehat{U} \cdot (\delta + \Theta_\ell - \sigma_{tot}).$$

By the selection of Θ_ℓ,

$$M_\ell^*(\delta) + (m-a) \cdot (E_\ell^*(\lambda) - 1) + (a-1) \cdot (\gamma_\ell^u(\lambda) + C_\ell + \max(0, \gamma_\ell^u(\lambda-1) - 1))$$
$$\geq \widehat{U} \cdot (\delta + x + \gamma_\ell^u(K_\ell) + C_\ell - \sigma_{tot}).$$

Setting the bounds on $E_\ell^*(\lambda)$ and $M_\ell^*(\delta)$ given by Lemmas 5.18 and 5.19 into the inequality above, we have

$$U_{sum} \cdot \delta + U(m-1) \cdot x + \mathcal{W} + \sum_{T_i \in \tau} L_i(C_{\ell,i}) + (m-a) \cdot (Y_\ell(\lambda) + u_\ell \cdot x - 1)$$
$$+ (a-1) \cdot (\gamma_\ell^u(\lambda) + C_\ell + \max(0, \gamma_\ell^u(\lambda-1) - 1))$$
$$\geq \widehat{U} \cdot (\delta + x + \gamma_\ell^u(K_\ell) + C_\ell - \sigma_{tot}).$$

Because $U_{sum} \leq \widehat{U}$ by the statement of the theorem and $\delta \geq 0$, we have

$$U(m-1) \cdot x + \mathcal{W} + \sum_{T_i \in \tau} L_i(C_{\ell,i}) + (m-a) \cdot (Y_\ell(\lambda) + u_\ell \cdot x - 1)$$
$$+ (a-1) \cdot (\gamma_\ell^u(\lambda) + C_\ell + \max(0, \gamma_\ell^u(\lambda-1) - 1))$$

140

$$\geq \widehat{U} \cdot (x + \gamma_\ell^u(K_\ell) + C_\ell - \sigma_{tot}).$$

After regrouping, we have

$$\mathcal{W} + \sum_{T_i \in \tau} L_i(C_{\ell,i}) + (m-a) \cdot (Y_\ell(\lambda) - 1)$$

$$+ (a-1) \cdot (\gamma_\ell^u(\lambda) + C_\ell + \max(0, \gamma_\ell^u(\lambda - 1) - 1)) - \widehat{U} \cdot (\gamma_\ell^u(K_\ell) + C_\ell - \sigma_{tot})$$

$$\geq x \cdot (\widehat{U} - (m - a) \cdot u_\ell - U(m - 1)).$$

Solving the above inequality for x, we have

$$x \leq \frac{\mathcal{W} + \widehat{U} \cdot \sigma_{tot} + V_\ell(\lambda) + \sum_{T_i \in \tau} L_i(C_{\ell,i})}{\widehat{U} - (m - a) \cdot u_\ell - U(m - 1)}, \tag{5.39}$$

where $V_\ell(\lambda) = (m - a) \cdot (Y_\ell - 1) + (a - 1) \cdot (\gamma_\ell^u(\lambda) + C_\ell + \max(0, \gamma_\ell^u(\lambda - 1) - 1) - \widehat{U} \cdot (\gamma_\ell^u(K_\ell) + C_\ell)$. From Definition 5.33, we have $m - a \geq 0$ and $a \geq 1$. Thus, since the function $\gamma_h^u(k)$ is non-decreasing, $V_h(k) \leq V_h$, where V_h is defined in the statement of the theorem. Maximizing the right-hand side of (5.39) by task T_ℓ, we have

$$x \leq \max_{T_h \in \tau} \left(\frac{\mathcal{W} + \widehat{U} \cdot \sigma_{tot} + V_h + \sum_{T_i \in \tau} L_i(C_{h,i})}{\widehat{U} - (m - a) \cdot u_h - U(m - 1)} \right).$$

This contradicts (5.37). $\qquad\square$

The result of Theorem 5.7 is closely related to the results of Devi (2006) and Theorem 3.1 in Chapter 3. In particular, the requirement $\widehat{U} - (m - a) \cdot \max(u_i) - U(m - 1)$ to be positive is a sufficient condition for maximum job response times (deadline tardiness) to be bounded.

5.8 Multiprocessor Analysis: A Case Study

Our analysis can be used to derive response-time bounds for workloads that partitioning schemes cannot accommodate and for workloads that cannot be efficiently analyzed under the widely-studied periodic and sporadic models. To illustrate this, we applied our analysis to a part of the MPEG-2 video decoder application presented in Example 1.5 in Section 1.6.

Experimental setup. In our experiments, we considered two variants of the previously-studied system shown in Figure 4.11(a) in which PE1 is a three-processor system running four identical VLD+IQ tasks,
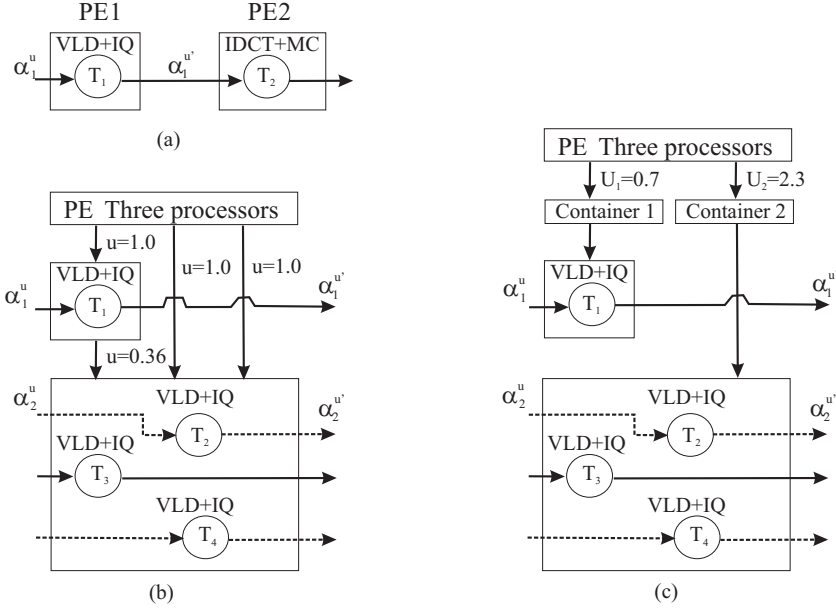
Figure 5.6: **(a)** A video-processing application. Experimental setup **(b)** without and **(c)** with containers.

T_1, T_2, T_3, and T_4. The two modified systems are illustrated in insets (b) and (c) of Figure 4.11 and explained in further detail below. For conciseness, we refer to the systems in these three insets as the (a)-, (b)-, and (c)-systems, respectively. To assess the usefulness of our analysis, we computed output curves of the four tasks so that they can be used in further analysis. We assumed zero scheduling and system overheads (the inclusion of such overheads in our analysis is beyond the scope of this dissertation).

The goal of our experiments was to compare different ways of implementing and analyzing the (b)- and (c)-systems. As we shall see, both systems can be implemented on three processors if global scheduling is used; in this case, they can be analyzed using the techniques of chapter but not using prior global schedulability analysis methods. Moreover, if the system is instead partitioned (allowing uniprocessor real-time calculus to be applied on each processor), then four processors are required.

In the analysis, we used a trace of 6×10^5 macroblock processing events obtained in prior work for the VLD+IQ task during a simulation of the (a)-system using a SimpleScalar architecture (Chakraborty et al., 2006; Phan et al., 2008).

We first determined execution times of macroblock instructions by examining a repeating pattern of 228,096 consecutive macroblock instruction lengths in the middle of the trace and assuming a 500 MHz processor frequency. We found that all macroblock processing times in the trace are under $164\mu s$ and the best-case macroblock processing time is $2\mu s$. These values are comparable to characteristic preemption and migration costs for multiprocessor systems measured in recent studies for architectures with higher

142

processor frequencies (Brandenburg et al., 2008a; Brandenburg and Anderson, 2009). Therefore it is not practical to invoke a job for each arriving macroblock. We thus assumed that a job is invoked for processing a single frame, which consists of 1,584 macroblocks, and obtained $\gamma_i^u(k)$ and $\gamma_i^l(k)$ as in Definition 5.1 for frames. We found that all frame processing times in the trace were under $\gamma_i^u(1) = 70ms$, which we set to be the maximum job execution time (the best-case execution time is $\gamma_i^l(1) = 18ms$). The function $\alpha_i^u(\Delta)$ in Definition 5.2 was obtained by examining macroblock and frame arrival times. We computed $\mathcal{A}_i^{-1}(k)$ in Definition 5.3 as well as linear bounds for $\alpha_i^u(\Delta)$ and $\gamma_i^u(k)$ as in (5.2) and (5.3) using the RTC Toolbox (Wandeler and Thiele, 2006).

In the (b)- and (c)-systems, three fully-available processors are used for scheduling tasks $T_1, \ldots, T_4$. However the scheduling algorithms in these two systems are different.

In the (b)-system, task T_1 is statically prioritized over the other tasks. In such a system, task T_1 can process a time-critical video stream and tasks T_2, T_3, and T_4 can process low-priority video streams. The remaining tasks T_2, T_3, and T_4 are scheduled by GEDF using the supply from two fully-available processors and that remaining on a third processor after accommodating task T_1. In Figure 4.11(b), down arrows are used to depict the long-term available utilization on each processor.

In the (c)-system, task T_1 and tasks $T_2, \ldots, T_4$ are encapsulated into two containers C_1 and C_2, respectively, as shown in Figure 4.11(c). The available processor time is distributed among these two containers as follows. Two processors are dedicated for scheduling tasks in C_2. The time on the third processor is allocated using periodic server tasks S_1 and S_2 with execution times e_1 and e_2 and periods p_1 and p_2. The jobs of S_1 and S_2 are scheduled using uniprocessor EDF. When task S_1 is scheduled, a job of T_1 is scheduled. Tasks $T_2, \ldots, T_4$ are scheduled by GEDF using the supply from two fully-available processor and the time available on the third processor when S_2 is scheduled. To ensure schedulability of the underlying tasks, the execution times and periods for server tasks should be selected as follows. $e_1/p_1 = \widehat{U_1} \geq u_1$, $2 + e_2/p_2 = \widehat{U_2} \geq u_2 + u_3 + u_4$, and $e_1/p_1 + e_2/p_2 = 1$. In Figure 4.11(c), down arrows to the container boxes denote the long-term guaranteed utilization in the respective container. The scheme described above is an application of the hierarchical scheduling scheme introduced in Chapter 4. In contrast to the (b)-system, in the (c)-system, task T_1 is temporally isolated from the other tasks.

Results. To show that existing analysis techniques are inapplicable or are too pessimistic in the given setup, some of the properties of the input streams and the VLD+IQ task need to be emphasized.

First, for both (b)- and (c)-systems, the minimum job inter-arrival time is $18ms$. Because the long-term arrival rate is $R_i = 0.025$, the arriving stream cannot be re-shaped to achieve a minimum job inter-arrival time greater than $p_i = 1/R_i = 40ms$ so that the long-term arrival rate is preserved.

Second, while the long-term worst-case execution time is $\overline{e_i} = 25.57ms$ (see Definition 5.1 and (5.3)), the maximum processing time of a single frame is $70ms$, so assuming that each job executes for its worst-case execution time would result in heavy overprovisioning. The long-term per-task utilization is $u_i = R_i \cdot \overline{e_i} = 0.025 \cdot 25.57 = 0.64$. Finally, the total utilization is $U = \sum_{i=1}^{4} u_i = 2.56$. Therefore, the task set $\{T_1, \ldots, T_4\}$ cannot be partitioned onto three processors (four processors are needed, actually), so global scheduling is required.

Because the worst-case job execution time is $e_i^{\max} = \gamma_i^u(1) = 70ms$ and the minimum job inter-arrival time is $p_i = 18ms$, we have $e_i^{\max}/p_i = 3.88 > 1$. Therefore, both (b)- and (c)-systems *cannot be analyzed using prior results for periodic and sporadic task models*, which require $p_i > 0$ and $e_i^{\max}/p_i \leq 1$.

Figure 5.7 depicts the job completion curve $\alpha_1^{u\prime}$ for task T_1 in the (a)- and (b)-systems, the curve $\alpha_2^{u\prime}$ for task T_2 in the (b)-system, and the input curve α_1^u. (Note that, in the (b)- and (c)-systems, tasks $T_1, \ldots, T_4$ have the same input curve α_1^u, and the completion curves for $T_2, \ldots, T_4$ are the same (within the respective system).) Figure 5.8 shows the input and completion curves for tasks T_1 and T_2 in the (c)-system.

Because task T_1 is effectively scheduled on a dedicated processor in the (a)- and (b)-systems, the output curves for T_1 in these two systems were obtained using prior results in real-time calculus for uniprocessor systems.

For the (b)-system, we calculated the maximum response time for T_1 and then applied Theorem 5.2 to find the supply available to tasks T_2, T_3, and T_4. We then calculated their response-time bounds $\Theta_i^\prime$ using Theorem 5.7. After that, we set $\Theta_i = \lfloor \Theta_i^\prime \cdot 0.83 \rfloor = 989ms$, which is the minimum value such that the conclusion of Corollary 5.3 still holds. The multiplier 0.83 was found by running a binary search procedure. We then computed completion curves using Theorem 5.1.

For the (c)-system, we constructed two periodic server tasks S_1 and S_2 with execution times $7ms$ and $3ms$, respectively, and period $10ms$. These values for execution times and periods are the smallest multiples of a typical quantum length of $1ms$ that give an approximate task utilization of 0.64. We used prior results to calculate the guaranteed processor time to each of the containers C_1 and C_2 (Leontyev and Anderson, 2009).

Because, in the (c)-system, task T_1 is effectively scheduled on one processor with limited availability, we calculated the output curve for task T_1 using prior results in uniprocessor real-time calculus. Given the supply guaranteed to container C_2, we calculated for tasks T_2, T_3, and T_4 the response-time bound $\Theta_i = 949ms$ and the completion curves similarly to those in the (b)-system.

The resulting curves have the same long-term completion rate in all the three systems. Task T_1 has
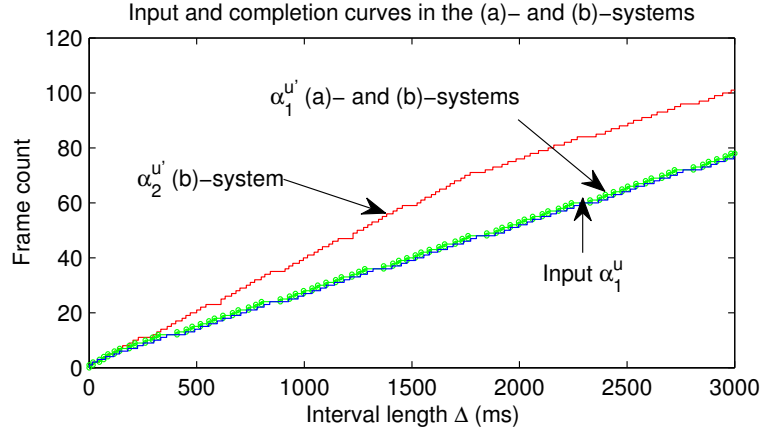
144

Figure 5.7: Job arrival curve α^u and completion curves $\alpha^{u\prime}$ for tasks T_1 and T_2 in the (a)- and (b)-systems.
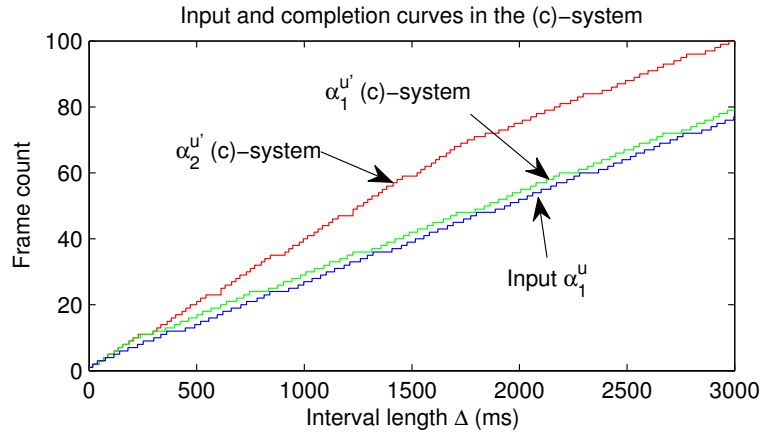
Figure 5.8: Job arrival curve α^u and completion curves $\alpha^{u\prime}$ for tasks T_1 and T_2 in the (c)-system.

the shortest possible maximum response time in both the (a)- and (b)-systems. However, the large job response times of tasks $T_2, \ldots, T_4$ in the (b)- and (c)-systems cause a larger degree of burstiness in the output event streams. This burstiness is a result of conservatism in the analysis.

Overall, the (b)- and (c)-systems have the advantage of needing only *three* processors to accommodate four video streams, while with partitioned scheduling, *four* dedicated processors are required. This advantage in the number of processors comes at the expense of larger buffers for storing partially decoded macroblocks for tasks T_2, T_3, and T_4. (The additional buffer size is the maximum difference between the output curves $\alpha_1^{u\prime}$ in the (b)- and (c)- and (a)- systems.) The output buffer for tasks $T_2, \ldots, T_4$ should be at least 25 additional frames, which is 1 second worth of video.

We conclude this section with a few comments about the running time of the analysis procedures. We have implemented these procedures as a set of MATLAB functions extending the RTC Toolbox. Though the procedure presented in Section 5.4 has pseudo-polynomial time complexity (like many other

schedulability tests presented elsewhere), the time needed to verify response times using Corollary 5.1 can be large, especially for complex arrival and execution-time patterns. In our experimental study of the (b)- and (c)-systems, we found that the required response-time bounds could be calculated in about a couple of minutes, by using Theorem 5.7 to obtain initial bounds, which were then refined using Corollary 5.1 (on a 1.7 GHz single-processor desktop system).

5.9   Summary

In this chapter, we have studied a multiprocessor PE, where (partially available) processors are managed by a global scheduling algorithm and jobs are triggered by streams of external events. This work is of importance because it allows workloads to be analyzed for which existing schedulability analysis methods are completely inapplicable (e.g., the system cannot be described efficiently using conventional periodic/sporadic task models) and for which partitioning techniques are unnecessarily restrictive.

The research in this dissertation is part of a broader effort, the goal of which is to produce a practical compositional framework, based on real-time calculus, for analyzing multiprocessor real-time systems. Towards this goal, the contributions of this chapter are as follows. We designed a pseudo-polynomial-time procedure that can be used to test whether job response times occur within specified bounds. Given these bounds, we computed upper and lower bounds on the number of job completion events over any interval of length Δ and a lower bound on the supply available after scheduling all incoming jobs. These bounds can be used as inputs for other PEs thereby resulting in a compositional analysis framework.

A number of unresolved issues of practical importance remain. First, *efficient* methods are needed for determining response-time bounds when they are not specified — this is probably the most important unresolved issue left. As a partial solution, we provided closed-form expressions for computing response-time bounds, but we do not know how pessimistic they are. Second, the schedulability test itself could possibly be improved by incorporating information about lower bounds on job arrivals and execution times and upper bounds on supply. Third, real-time interfaces as in (Chakraborty et al., 2006) need to be derived for the multiprocessor case to achieve full compatibility with uniprocessor real-time calculus. Fourth, the inherent pessimism introduced by applying real-time calculus methods on multiprocessors needs to be assessed.

Chapter 6

Conclusion and Future Work

In this dissertation, we extended prior work on multiprocessor soft real-time scheduling to enable the analysis of component-based systems, specifically by introducing extensions to real-time calculus and a novel scheme for scheduling real-time containers on a multiprocessor.

Prior work on real-time calculus did not consider resource-efficient global schedulers, and prior work in the area of global multiprocessor scheduling mostly considered workloads consisting of independent sporadic tasks. In Chapters 4 and 5, we have presented the analysis of workloads described by sporadic and streaming task models scheduled using global scheduling algorithms on a multiprocessor with potentially restricted supply. These techniques significantly extend the assortment of building blocks to be used for the design and analysis of embedded and distributed systems in addition to those offered by conventional real-time calculus.

As multicore platforms have become standard within many domains, creating resource-efficient scheduling policies and analysis methods for such platforms has become necessary to ensure provably acceptable system performance.

6.1 Summary of Results

In Chapter 1, we formulated the thesis statement given below, which was to be supported by this dissertation.

With the exception of static-priority algorithms, virtually all previously studied global real-time scheduling algorithms ensure bounded deadline tardiness for implicit-deadline sporadic task systems. This property is preserved even if the processing capacity of some processors is not fully available, provided that the long-term execution demand does not exceed the total available processing capacity. Well-studied global

schedulers such as GEDF and FIFO ensure bounded maximum response times in systems with complex job arrival and execution patterns as described by the streaming task model. The use of such algorithms enables component-based systems with predominantly soft timing constraints to be built while incurring little or no utilization loss in settings where partitioning approaches are too costly in terms of needed processing resources.

In support to this thesis statement, in Chapter 3, we have presented a general tardiness-bound derivation that applies to a wide variety of global scheduling algorithms for sporadic tasks. Our results show that, with the exception of static-priority algorithms, most global algorithms of interest in the real-time-systems community have bounded tardiness. When considering new algorithms, the question of whether tardiness is bounded can be answered in the affirmative by simply showing that the required prioritization can be specified. Bounded tardiness is preserved even if the capacity of each processor that is available to the (soft) real-time workload is restricted (provided that the entire system is not overloaded and maximum per-task utilizations are not too high).

Using these results about bounded tardiness on restricted-capacity platforms, in Sections 4.4 and 4.4.1, we have identified conditions under which a restricted-capacity platform can be fully utilized without constraining the maximum per-task utilization. These observations led to the development of a multi-processor bandwidth-reservation scheme for hierarchically organized real-time containers in Chapter 4. Under this scheme each real-time container can reserve any fraction of processor time (even the capacity of several processors) to schedule its children. The presented scheme provides temporal isolation among containers so that each container can be analyzed separately. Our scheme is novel in that soft real-time components incur no utilization loss. This stands in sharp contrast to hierarchical schemes for hard (only) real-time systems, where the loss per level can be so significant, arbitrarily deep hierarchies simply become untenable.

Finally, understanding the behavior of soft real-time tasks on a globally-scheduled multiprocessor is essential for the analysis of more sophisticated workloads. In Chapter 5, we have proposed a framework for the analysis of multiprocessor processing elements with streaming tasks where the constituent processors are managed according to a global multiprocessor scheduling algorithm. Such processing elements can be used for building complex applications that cannot be analyzed using state-of-the-art multiprocessor scheduling techniques, and that must be overprovisioned, wasting processing resources, if analyzed using conventional real-time calculus. Sporadic and streaming task sets under GEDF, and static-priority schedulers, can be analyzed in this framework. We showed its viability in a case study considering a realistic multimedia application.

6.2 Other Contributions

In this section, we briefly discuss other contributions by the author to the field of real-time systems that are outside of the scope of this dissertation.

Multiprocessor scheduling on asymmetric platforms. In (Leontyev and Anderson, 2007b), we proposed an approach for supporting sporadic soft real-time tasks running on an asymmetric multicore platform. In such a platform, multiple processing cores are placed on one chip or several chips, and all processing cores have same instruction set, but potentially different performance levels. As a result, tasks can have different execution times when running on different types of cores. The usage of such a platform can be beneficial if there is a need to accommodate both parallelizable and inherently-sequential applications on the same platform.

In our work, we have presented a new algorithm, EDF-ms (EDF for <u>m</u>ulti-<u>s</u>peed platforms), which can be used for scheduling sporadic soft real-time task systems on asymmetric multicore platforms. To our knowledge, our work is the first to propose a scheduling approach for such heterogeneous platforms that is suitable for *soft* real-time workloads that require bounded deadline tardiness. Our algorithm is capable of fully utilizing the processing capacity of the system, provided certain very slight restrictions on task utilizations hold. This property comes at the price of needing to migrate tasks, as required in global scheduling approaches such as GEDF.

Unified schedulability test for GEDF. In (Leontyev and Anderson, 2008b), we proposed a schedulability test for the sporadic task model under preemptive and non-preemptive global EDF that treats hard and soft real-time constraints uniformly. Particularly, each task T_i has a specified tardiness bound $\Theta_i \geq 0$ so the test checks whether these bounds are met. The results presented in (Leontyev and Anderson, 2008b) are closely related to those in this dissertation in Chapter 5 except that here we have examined more general task and supply models.

Real-time synchronization protocols. The author participated in several group efforts that were more implementation-oriented and were led by other researchers. Such efforts included a series of papers regarding real-time synchronization protocols. In (Devi et al., 2006), the hard and soft real-time schedulability of sporadic task sets using spin-lock-protected and lock-free shared objects for synchronization was studied. It was shown that using non-preemptive queue locks results in better schedulability. In (Brandenburg et al., 2008b), blocking and non-blocking approaches to sharing objects among real-time tasks were compared. The authors implemented spin-lock-protected, lock-free, and wait-free variants of

several classic data structures and measured access times under different conditions. These access times were later used in the schedulability analysis of randomly generated task sets. In (Block et al., 2007), the authors proposed a new *Flexible Multiprocessor Locking Protocol* (FMLP), which is a hybrid blocking synchronization protocol that uses spin-locks to protect short critical sections and semaphores to protect long critical sections. Schedulability conditions for tasks using the FMLP under various scheduling algorithms were established.

LITMUS$^{\text{RT}}$. In order to better understand how various global scheduling algorithms behave in practice, our research group constructed LITMUS$^{\text{RT}}$ (LInux Testbed for MUltiprocessor Scheduling in Real-Time systems). LITMUS$^{\text{RT}}$ is an extension of the Linux kernel that allows Linux tasks to have timing constraints, be managed using a user-defined scheduling algorithm, and use state-of-the-art real-time synchronization mechanisms (Calandrino et al., 2006; Brandenburg et al., 2007). LITMUS$^{\text{RT}}$ adds a number of hooks into the original Linux scheduling code so that user-defined scheduling functions can be called. For each scheduling algorithm implemented in LITMUS$^{\text{RT}}$, these functions are bundled into distinct *plugins*, which can be switched at runtime. LITMUS$^{\text{RT}}$ provides implementations of preemptive and non-preemptive versions of GEDF, the PD^2 Pfair algorithm, and PEDF. Also, in LITMUS$^{\text{RT}}$, there are a number of tracing and debugging tools that facilitate the development of new plugins as well as a collection of probes for measuring various system and scheduling overheads.

Interrupt accounting schemes. The results concerning schedulability on multiprocessors from Chapter 3 have been used to design novel methods for accounting for interrupts. Arriving interrupts take processor time and are not subject to regular scheduling. Thus, they can affect the timeliness of other real-time tasks in a system. Though system designers attempt to make interrupts as short as possible, their presence has to be accounted for in schedulability analysis. One of the interrupt accounting schemes presented in (Brandenburg et al., 2009) subtracts the total interrupt processing time from the full processor supply and then treats soft real-time tasks as though they were running on a reduced-capacity platform. In (Brandenburg et al., 2009), this and other interrupt accounting methods and interrupt dispatching schemes are quantitatively evaluated using randomly generated task sets.

6.3   Future Work

There are several ways in which the work described in this dissertation could be extended, as we discuss next.

Deriving tight tardiness bounds and devising reactive tardiness-reduction techniques.   Our experimental results suggest that actual tardiness under EDZL is likely to be very low. It would be interesting to improve our analysis as it applies to EDZL in order to obtain a tight tardiness bound. Tardiness bounds for other algorithms, like GEDF, can likely be improved as well. Given that there is interest in the Linux community for supporting similar scheduling algorithms, such theoretical work would provide a solid foundation to support this choice. It would also be interesting to investigate reactive techniques that could be applied at runtime to lessen tardiness for certain jobs by redefining priority points, as circumstances warrant. Such techniques might exploit the fact that our framework allows priority definitions to be changed rather arbitrarily at runtime.

Introducing dynamic containers.   An important topic for future work is to enable dynamic container creation and the joining/leaving of tasks. To achieve this goal, recent results on changing task parameters such as execution times and periods at runtime could be helpful (Block et al., 2008). These results show that, if bounded tardiness has to be supported, then tasks cannot change their parameters at arbitrary times. Similar restrictions could pertain to containers if tasks are allowed to change their parameters or migrate between containers. Also of importance is the inclusion of support for synchronization. To implement shared objects, one needs to consider non-blocking synchronization protocols in addition to lock-based alternatives. Finally, overheads need to be measured for an implementation of the hierarchical scheduling framework within LITMUS$^{\text{RT}}$.

Improving multiprocessor real-time calculus.   The extensions to real-time calculus that we presented could be further extended in several directions. First, the inherent pessimism introduced by applying real-time calculus methods on multiprocessors needs to be assessed. Second, methods with low computational complexity are needed for determining response-time bounds when they are not specified. As a partial solution, in Section 5.7, we provided closed-form expressions for computing response-time bounds, but we do not know how pessimistic they are. Also, these closed-form expressions are applicable only to EDF-like schedulers so similar bounds have to be derived for static-priority and unrestricted dynamic-priority schedulers as well. Third, the schedulability test itself could possibly be improved by incorporating information about lower bounds on job arrivals and execution times and upper bounds on supply. Fourth, real-time interfaces as in (Chakraborty et al., 2006) need to be derived for the multiprocessor case to achieve full compatibility with uniprocessor real-time calculus.

Appendix A

Proofs for Lemmas in Chapter 3

The following claim is used in proving Lemma 3.6 and Lemma A.1.

Claim A.1.

(a) *If, for job $T_{i,g}$, $r_{i,g} \geq t$, then $\mathsf{A}(T_{i,j}, 0, t, \mathcal{PS}) = 0$ for each $j \geq g$.*

(b) *If, for job $T_{i,g}$, $r_{i,g} < t \leq d_{i,g}$, then $\mathsf{A}(T_{i,j}, 0, t, \mathcal{PS}) = 0$ for each $j > g$.*

Proof. (a) follows from the fact that no job $T_{i,j}$ such that $r_{i,j} \geq t$ receives an allocation before its release time in the PS schedule $\mathcal{PS}$. If $r_{i,g} < t \leq d_{i,g}$, then $j > g$ implies that $r_{i,j} \geq r_{i,g} + p_i = d_{i,g} \geq t$, which, by (a), implies (b). □

Lemma 3.6: $\mathsf{lag}(T_k, t, \mathcal{S}) \leq x \cdot u_k + e_k$ *for any task T_k and $t \in [0, t_d]$.*

Proof. Let $d_{k,j}$ be the deadline of the earliest pending job of T_k, $T_{k,j}$, in the schedule $\mathcal{S}$ at time t. Let $\gamma_{k,j} < e_{k,j}$ be the amount of time for which $T_{k,j}$ executes before t in the schedule $\mathcal{S}$. By (3.5) and the selection of $T_{k,j}$,

$$
\begin{aligned}
\mathsf{lag}(T_k, t, \mathcal{S}) &= \sum_{h \geq 1} \mathsf{lag}(T_{k,h}, t, \mathcal{S}) \\
&= \sum_{h \geq j} \mathsf{lag}(T_{k,h}, t, \mathcal{S}) \\
&= \sum_{h \geq j} (\mathsf{A}(T_{k,h}, 0, t, \mathcal{PS}) - \mathsf{A}(T_{k,h}, 0, t, \mathcal{S})) \\
&= \mathsf{A}(T_{k,j}, 0, t, \mathcal{PS}) - \mathsf{A}(T_{k,j}, 0, t, \mathcal{S}) + \sum_{h > j} \mathsf{A}(T_{k,h}, 0, t, \mathcal{PS}) - \sum_{h > j} \mathsf{A}(T_{k,h}, 0, t, \mathcal{S}). \quad \text{(A.1)}
\end{aligned}
$$

We now bound each term in the equation above. Since the earliest pending job $T_{k,j}$ executes for $\gamma_{k,j}$ time units before time t in the schedule $\mathcal{S}$,

$$
\mathsf{A}(T_{k,j}, 0, t, \mathcal{S}) = \gamma_{k,j} \quad \text{and} \quad \sum_{h > j} \mathsf{A}(T_{k,h}, 0, t, \mathcal{S}) = 0. \quad \text{(A.2)}
$$

Bounds for the remaining terms depend on the relationship between $d_{k,j}$ and t.

Case 1: $d_{k,j} < t$. Since $T_{k,j}$ does not execute before its release time and finishes at $d_{k,j}$ in $\mathcal{PS}$, from the condition of Case 1, it follows that

$$\mathsf{A}(T_{k,j}, 0, t, \mathcal{PS}) = \mathsf{A}(T_{k,j}, r_{k,j}, d_{k,j}, \mathcal{PS}) = e_{k,j}. \tag{A.3}$$

Since the job $T_{k,j+1}$ cannot commence execution in $\mathcal{PS}$ earlier than time $d_{k,j}$,

$$\sum_{h>j} \mathsf{A}(T_{k,h}, 0, t, \mathcal{PS}) \leq u_k \cdot (t - d_{k,j}). \tag{A.4}$$

Setting (A.2), (A.3), and (A.4) into (A.1), we get

$$\mathsf{lag}(T_k, t, \mathcal{S}) \leq e_{k,j} - \gamma_{k,j} + u_k \cdot (t - d_{k,j}). \tag{A.5}$$

Because $d_{k,j} < t \leq t_d$ holds, by Property (P), $T_{k,j}$ has tardiness at most $x + e_k$. Let $compl(T_{k,j}, t)$ be the length of the interval after time t where $T_{k,j}$ is pending. Then, $t + compl(T_{k,j}, t) \leq d_{k,j} + x + e_k$, and hence,

$$t - d_{k,j} \leq x + e_k - compl(T_{k,j}, t). \tag{A.6}$$

Because $T_{k,j}$ executes for $\gamma_{k,j}$ time units before time t, $compl(T_{k,j}, t) \geq e_{k,j} - \gamma_{k,j}$. Setting the last inequality into (A.6), we get $t - d_{i,j} \leq x + e_k - e_{k,j} + \gamma_{k,j}$. From (A.5), we therefore have

$$
\begin{aligned}
\mathsf{lag}(T_k, t, \mathcal{S}) &\leq e_{k,j} - \gamma_{k,j} + u_k \cdot (t - d_{k,j}) \\
&\leq e_{k,j} - \gamma_{k,j} + u_k \cdot (x + e_k - e_{k,j} + \gamma_{k,j}) \\
&= e_{k,j} + u_k \cdot x + \gamma_{k,j} \cdot (u_k - 1) + u_k \cdot (e_k - e_{k,j}) \\
&\leq u_k \cdot x + e_{k,j} + u_k \cdot (e_k - e_{k,j}) \\
&= u_k \cdot x + e_{k,j} \cdot (1 - u_k) + u_k \cdot e_k \\
&\quad \{\text{maximized if } e_{k,j} = e_k\} \\
&\leq u_k \cdot x + e_k.
\end{aligned}
$$

Case 2: $d_{k,j} \geq t$. In this case,

$$\mathsf{A}(T_{k,j}, 0, t, \mathcal{PS}) = \mathsf{A}(T_{k,j}, r_{k,j}, t, \mathcal{PS}) \leq u_{k,j} \cdot (t - r_{k,j}) \leq u_k \cdot (d_{k,j} - r_{k,j}) = u_k \cdot p_k = e_k. \tag{A.7}$$

By the condition of Case 2, for any job $T_{k,h}$ such that $h > j$, $r_{k,h} \geq t$ holds, and hence, by Claim A.1,

$$\sum_{h>j} \mathsf{A}(T_{k,h}, 0, t, \mathcal{PS}) = 0. \tag{A.8}$$

Setting (A.2), (A.7), and (A.8) into (A.1) we get

$$\mathsf{lag}(T_k, t, \mathcal{S}) \leq e_{k,j} - \gamma_{k,j} \leq e_k + u_k \cdot x,$$

where the latter inequality trivially follows, since $x \geq \rho \geq 0$ (see (P)). The lemma follows. $\qquad\square$

Lemma 3.7: $\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) \leq \mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i + \sum_{k=1}^m \widehat{u_k} \cdot \sigma_k$.

Proof. By (3.7),

$$\mathsf{LAG}(\mathbf{d}, t_d, \mathcal{S}) = \mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) + \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{S}). \tag{A.9}$$

To compute $\mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{S})$, we split $[t_n, t_d)$ into b non-overlapping intervals $[t_{p_s}, t_{q_s})$, $1 \leq s \leq b$, such that $t_n = t_{p_1}$, $t_{q_{s-1}} = t_{p_s}$, and $t_{q_b} = t_d$. These intervals are defined so that, for each interval $[t_{p_s}, t_{q_s})$, if processor h is unavailable at time $t \in [t_{p_s}, t_{q_s})$, then it is unavailable throughout the entire interval $[t_{p_s}, t_{q_s})$. We further assume that each interval $[t_{p_s}, t_{q_s})$ is defined so that if a job $T_{k,j}$ executes at some point in the interval in schedule $\mathcal{S}$, then it executes continuously throughout the interval in $\mathcal{S}$. Note that such a job $T_{k,j}$ does not necessarily execute continuously throughout $[t_n, t_d)$. The allocation difference for $\mathbf{d}$ throughout the interval $[t_n, t_d)$ is thus

$$\mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{S}) = \sum_{s=1}^b \left(\mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{S}) \right).$$

We now bound the allocation difference in the PS schedule $\mathcal{PS}$ and the schedule $\mathcal{S}$ across each of the intervals $[t_{p_s}, t_{q_s})$. The sum of these bounds gives us a bound on the total allocation difference throughout $[t_n, t_d)$. By the definition of a PS schedule,

$$\mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{PS}) \leq U_{sum} \cdot (t_{q_s} - t_{p_s}). \tag{A.10}$$

For each interval $[t_{p_s}, t_{q_s})$, we let $\alpha_s \subseteq \tau_{\mathbf{DH}}$ denote those tasks that execute their jobs in $\mathbf{DH}$ continuously throughout $[t_{p_s}, t_{q_s})$ in the schedule $\mathcal{S}$. Due to selection of t_n, within each interval $[t_{p_s}, t_{q_s})$ in schedule $\mathcal{S}$ two alternatives are possible:

1. m available processors are occupied by tasks with ready jobs in $\mathbf{d}$.

2. Some tasks with ready jobs in $\mathbf{d}$ do not execute because some processors are unavailable and/or other available processors execute tasks in α_s. (Note that, by Lemma 3.5, jobs in **DLH** and **DLL** cannot execute at time instants when there are ready unscheduled jobs in $\mathbf{d}$.)

For each interval $[t_{p_s}, t_{q_s})$, we define κ_s to be the number of unavailable processors in that interval. The number of available processors in $[t_{p_s}, t_{q_s})$ is thus $m - \kappa_s$. Therefore,

$$\mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{S}) = (t_{q_s} - t_{p_s}) \cdot (m - |\alpha_s| - \kappa_s)$$

$$= -(t_{q_s} - t_{p_s}) \cdot |\alpha_s| + (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s). \tag{A.11}$$

Subtracting (A.11) from (A.10), we get

$$\mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_{p_s}, t_{q_s}, \mathcal{S})$$

$$\leq (t_{q_s} - t_{p_s}) \cdot U_{sum} - (-(t_{q_s} - t_{p_s}) \cdot |\alpha_s| + (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s))$$

$$= (t_{q_s} - t_{p_s}) \cdot U_{sum} + (t_{q_s} - t_{p_s}) \cdot |\alpha_s| - (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s)$$

$$= (t_{q_s} - t_{p_s}) \cdot U_{sum} + (t_{q_s} - t_{p_s}) \cdot \sum_{T_i \in \alpha_s} 1 - (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s). \tag{A.12}$$

Summing (A.12) over all intervals $[t_{p_s}, t_{q_s})$, we have

$$\mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{S})$$

$$\leq \sum_{s=1}^{b} (t_{q_s} - t_{p_s}) \cdot U_{sum} + \sum_{s=1}^{b} \sum_{T_i \in \alpha_s} (t_{p_s} - t_{q_s}) - \sum_{s=1}^{b} (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s)$$

$$= (t_d - t_n) \cdot U_{sum} + \sum_{s=1}^{b} \sum_{T_i \in \alpha_s} (t_{p_s} - t_{q_s}) - \sum_{s=1}^{b} (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s). \tag{A.13}$$

For each task $T_i \in \tau_{\mathbf{DH}}$, the sum of the lengths of the intervals $[t_{p_s}, t_{q_s})$, in which jobs of T_i from **DH** execute continuously before time t_d is at most δ_i (see Definition 3.9). Thus,

$$\sum_{s=1}^{b} \sum_{T_i \in \alpha_s} (t_{p_s} - t_{q_s}) \leq \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i. \tag{A.14}$$

Now consider $\sum_{s=1}^{b} (t_{q_s} - t_{p_s}) \cdot (m - \kappa_s)$. Since κ_s is the number of unavailable processors within the interval $[t_{p_s}, t_{q_s})$, $(m - \kappa_s) \cdot (t_{q_s} - t_{p_s})$ is the amount of processor time available to tasks in τ

within $[t_{p_s}, t_{q_s})$. The sum of these times for all the intervals $[t_{p_s}, t_{q_s})$ is at least the total processor time guaranteed within $[t_n, t_d)$, because each processor is either unavailable or executes a task from τ within $[t_{p_s}, t_{q_s})$. Thus,

$$\sum_{s=1}^{b}(m - \kappa_s) \cdot (t_{q_s} - t_{p_s}) \geq \sum_{k=1}^{m} \beta_k^l(t_d - t_n). \tag{A.15}$$

By (1.1) and (A.15), we have

$$\sum_{s=1}^{b}(m - \kappa_s) \cdot (t_{q_s} - t_{p_s}) \geq \sum_{k=1}^{m} \beta_k^l(t_d - t_n) \geq \sum_{k=1}^{m} \widehat{u_k} \cdot (t_d - t_n - \sigma_k). \tag{A.16}$$

Substituting (A.14) and (A.16) into (A.13), we have

$$\mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{PS}) - \mathsf{A}(\mathbf{d}, t_n, t_d, \mathcal{S}) \leq (t_d - t_n)U_{sum} + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i - \sum_{k=1}^{m} \widehat{u_k} \cdot (t_d - t_n - \sigma_k)$$

$$= (t_d - t_n)\left(U_{sum} - \sum_{k=1}^{m} \widehat{u_k}\right) + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i + \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k$$

$$\{ \text{ by } (3.1) \}$$

$$\leq \sum_{k=1}^{m} \widehat{u_k} \cdot \sigma_k + \sum_{T_i \in \tau_{\mathbf{DH}}} \delta_i. \tag{A.17}$$

By (A.17) and (A.9), the lemma follows. □

The following definition and Lemmas A.1 and A.2 and used in proving Lemma 3.8.

Definition A.1. Let $\xi = \{T_i :: \exists T_{i,j} \in \mathbf{d}$ such that $T_{i,j}$ is ready at t_n^- in schedule $\mathcal{S}\}$.

Lemma A.1. If $T_i \notin \xi$, then $\sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) \leq 0$.

Proof. Consider task $T_i \notin \xi$ at time instant t_n^-. Let $T_{i,g}$ be the latest job such that $r_{i,g} < t_n$. Then $t_n \leq r_{i,j}$ for each $j > g$. By Claim A.1 (b),

$$\sum_{T_{i,j} :: T_{i,j} \in \mathbf{d} \wedge j > g} \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) = 0. \tag{A.18}$$

Also, in the PS schedule $\mathcal{PS}$, $T_{i,g}$'s allocation cannot be larger than its actual execution time $e_{i,g}$.

$$\mathsf{A}(T_{i,g}, 0, t_n, \mathcal{PS}) \leq e_{i,g}. \tag{A.19}$$

Because $T_i \notin \xi$, all jobs $T_{i,j}$ such that $T_{i,j} \in \mathbf{d}$ and $j < g$ complete by time t_n^- in both schedules $\mathcal{S}$ and

$\mathcal{PS}$, and hence,

$$\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) = \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}) \text{ for each } j < g \text{ and } T_{i,j} \in \mathbf{d}. \tag{A.20}$$

Also, all jobs with eligibility times at most t_n, including job $T_{i,g}$, for which $\epsilon_{i,g} \leq r_{i,g} < t_n$, complete by t_n in schedule $\mathcal{S}$. We thus have

$$\mathsf{A}(T_{i,g}, 0, t_n, \mathcal{S}) = e_{i,g}. \tag{A.21}$$

By (3.6), we have

$$\sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) = \sum_{T_{i,j} \in \mathbf{d}} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}))$$

$$\{\text{by (A.20)}\}$$

$$= \sum_{T_{i,j} \,::\, T_{i,j} \in \mathbf{d} \wedge j \geq g} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}))$$

$$\{\text{by (A.19) and (A.21)}\}$$

$$\leq \sum_{T_{i,j} \,::\, T_{i,j} \in \mathbf{d} \wedge j > g} \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \sum_{T_{i,j} \,::\, T_{i,j} \in \mathbf{d} \wedge j > g} \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S})$$

$$\{\text{by (A.18)}\}$$

$$\leq - \sum_{T_{i,j} \,::\, T_{i,j} \in \mathbf{d} \wedge j > g} \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S})$$

$$\leq 0.$$

The lemma follows. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

Lemma A.2. *If $T_i \in \xi$, then $\sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) \leq \mathsf{lag}(T_i, t_n, \mathcal{S})$.*

Proof. Because $T_i \in \xi$, there exists a job $T_{i,g}$ such that $d_{i,g} \leq t_d$ and $T_{i,g}$ is pending at t_n^-. Because jobs of T_i execute sequentially, jobs of T_i with deadlines after $d_{i,g}$ do not execute before time t_n, and hence,

$$\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}) = 0 \text{ for each job } T_{i,j} \notin \mathbf{d}. \tag{A.22}$$

We therefore have,

$$\mathsf{lag}(T_i, t_n, \mathcal{S})$$

$$\{\text{by (3.5)}\}$$

$$= \sum_{j \geq 1} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}))$$

$$= \sum_{(j \geq 1) \wedge T_{i,j} \in \mathbf{d}} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS})$$

$$- \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S})) + \sum_{(j \geq 1) \wedge T_{i,j} \notin \mathbf{d}} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}))$$

$$\{\text{by } (3.6)\}$$

$$= \sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) + \sum_{T_{i,j} \notin \mathbf{d}} (\mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS}) - \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{S}))$$

$$\{\text{by } (A.22)\}$$

$$= \sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) + \sum_{T_{i,j} \notin \mathbf{d}} \mathsf{A}(T_{i,j}, 0, t_n, \mathcal{PS})$$

$$\geq \sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}). \hspace{4cm} \square$$

Lemma 3.8: $\mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) \leq E_L + x \cdot U_L$.

Proof. If $t_n = 0$, then $\mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S}) = 0$ and the lemma holds trivially, so assume that $t_n > 0$. By Definition 3.10 and Definition A.1, all tasks in ξ execute at t_n^-, and hence, $|\xi| \leq m - 1$. Therefore,

$$\mathsf{LAG}(\mathbf{d}, t_n, \mathcal{S})$$

$$\{\text{by } (3.6)\}$$

$$= \sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S})$$

$$= \sum_{T_i \in \xi} \sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S}) + \sum_{T_i \notin \xi} \sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S})$$

$$\{\text{by Lemma A.1}\}$$

$$\leq \sum_{T_i \in \xi} \sum_{T_{i,j} \in \mathbf{d}} \mathsf{lag}(T_{i,j}, t_n, \mathcal{S})$$

$$\{\text{by Lemma A.2}\}$$

$$\leq \sum_{T_i \in \xi} \mathsf{lag}(T_i, t_n, \mathcal{S})$$

$$\{\text{by Lemma 3.6}\}$$

$$\leq \sum_{T_i \in \xi} (x \cdot u_i + e_i)$$

$$\{\text{because } |\xi| \leq m - 1\}$$

$$\leq E_L + x \cdot U_L. \qquad \qquad \square$$

The following claim is used in proving Lemmas 3.10 and 3.11.

Claim A.2. *If* $T_{i,k} \in \mathbf{DH}$*, then* $\chi(T_{i,k}, t') \leq t_d + \psi_a$ *for some* $a \neq i$ *and time* t'*.*

Proof. If $T_{i,k} \in \mathbf{DH}$, then, by (3.12), there exists $T_{a,b} \in \mathbf{d}$ such that $a \neq i$ and $\neg\mathsf{LP}(T_{i,k}, T_{a,b})$ holds. By (3.10), there exists t' such that

$$\chi(T_{i,k}, t') \leq d_{a,b} + \psi_a$$

$$\{\text{because } T_{a,b} \in \mathbf{d}, \text{ by } (3.11), d_{a,b} \leq t_d \}$$

$$\leq t_d + \psi_a.$$

The claim follows. $\qquad \qquad \square$

Lemma 3.10. *If* $T_{i,k} \in \mathbf{d} \cup \mathbf{DH}$*, then* $r_{i,k} \leq t_d + \rho$*.*

Proof. Because sets $\mathbf{d}$ and $\mathbf{DH}$ are disjoint we consider two cases.

Case 1: $T_{i,k} \in \mathbf{d}$. In this case, $r_{i,k} \leq d_{i,k} \leq t_d \leq t_d + \rho$, since $\rho \geq 0$.

Case 2: $T_{i,k} \in \mathbf{DH}$. By the condition of Case 2 and Claim A.2, there exists $a \neq i$ and t' such that $\chi(T_{i,k}, t') \leq t_d + \psi_a$. We thus have, for time t',

$$r_{i,k}$$

$$\{\text{by } (3.3)\}$$

$$\leq \chi(T_{i,k}, t') + \phi_i$$

$$\leq t_d + \psi_a + \phi_i$$

$$\{\text{by } (3.9)\}$$

$$\leq t_d + \rho.$$

The lemma follows. $\qquad \qquad \square$

Lemma 3.11. *If* $T_{i,k} \in \mathbf{DLH}$*, then* $r_{i,k} \leq t_d + \rho + \mu$*.*

Proof. Suppose that $T_{i,k} \in \mathbf{DLH}$. Then, by (3.14), there exists $T_{a,b} \in \mathbf{DH}$ such that $a \neq i$ and $\neg\mathsf{LP}(T_{i,k}, T_{a,b})$ holds. The latter implies that $\chi(T_{i,k}, t') \leq d_{a,b} + \psi_a$ holds for some time t'. We thus have, for time t',

$$r_{i,k}$$

$$\{\text{by (3.3)}\}$$

$$\leq \chi(T_{i,k}, t') + \phi_i$$

$$\leq d_{a,b} + \psi_a + \phi_i$$

$$= r_{a,b} + p_a + \psi_a + \phi_i$$

$$\{\text{by (3.9)}\}$$

$$\leq r_{a,b} + \mu$$

$$\{\text{by Lemma 3.10}\}$$

$$\leq t_d + \rho + \mu. \qquad \square$$

Theorem 3.3. *If $\mathcal{A}$ is an eventually-monotonic scheduling algorithm and its prioritization functions are augmented as described above, then no job is preempted while executing in a non-preemptive region.*

Proof. Suppose, contrary to the statement of the theorem, that job $T_{k,h}$ begins executing a non-preemptive region at time t_1 and, while still within that region, is preempted at time t_p by job $T_{a,b}$ that is either ready but not scheduled at time t_p^- or becomes eligible at t_p. Because $T_{k,h}$ cannot be scheduled earlier than $\epsilon_{k,h}$, we have

$$\epsilon_{k,h} \leq t_1 < t_p^- < t_p. \tag{A.23}$$

According to the priority augmentation rules, $\chi(T_{a,b}, t_p) = \chi^A(T_{a,b}, t_p)$. Below, we show that either $\chi^A(T_{a,b}, t_p) > r_{k,h} - G = \chi(T_{k,h}, t_p)$ holds or the tie-breaking between jobs $T_{a,b}$ and $T_{k,h}$ at times t_p^- and t_p is not consistent, and hence, job $T_{a,b}$ cannot be scheduled at time t_p as assumed. Let

$$r_c = r_{k,h} - \mu - \gamma - p_{max} - M. \tag{A.24}$$

Two cases are possible, based upon the release time of $T_{a,b}$.

Case 1: $r_c \leq r_{a,b}$. In this case,

$$\chi^A(T_{a,b}, t_p)$$

$$\{\text{by (3.3)}\}$$

$$\geq r_{a,b} - \phi_a$$

$$\{\text{by the condition of Case 1}\}$$

$$\geq r_c - \phi_a$$

$$\{\text{by (A.24)}\}$$

$$= r_{k,h} - \mu - \gamma - p_{max} - M - \phi_a$$

$$\{\text{by Definition 3.19}\}$$

$$> r_{k,h} - G$$

Case 2: $r_{a,b} < r_c$. In this case, we can show that $d_{a,b} + M < \epsilon_{k,h}$ holds.

$$d_{a,b} + M = r_{a,b} + p_a + M$$

$$\{\text{by the condition of Case 2}\}$$

$$< r_c + p_a + M$$

$$\{\text{by (A.24)}\}$$

$$= r_{k,h} - \mu - \gamma - p_{max} - M + p_a + M$$

$$\leq r_{k,h} - \mu - \gamma$$

$$\{\text{by (3.40)}\}$$

$$\leq \epsilon_{k,h} - \mu$$

$$\{\text{because } \mu \geq 0 \text{ (see (3.9))}\}$$

$$\leq \epsilon_{k,h} \tag{A.25}$$

Two subcases are possible, depending on whether job $T_{a,b}$ is ready at time t_p^-.

Subcase 1: *$T_{a,b}$ is not ready at time t_p^-.* In this case, by the selection of $T_{a,b}$, it becomes eligible at t_p, and hence, by (A.23),

$$\epsilon_{k,h} < t_p = \epsilon_{a,b}. \tag{A.26}$$

We can lower-bound $\chi^A(T_{a,b}, t_p)$ as follows.

$$\chi^A(T_{a,b}, t_p)$$

$$\{\text{by } (3.3)\}$$

$$\geq r_{a,b} - \phi_a$$

$$\geq \epsilon_{a,b} - \phi_a$$

$$\{\text{by } (A.26)\}$$

$$> \epsilon_{k,h} - \phi_a$$

$$\{\text{by } (3.40)\}$$

$$\geq r_{k,h} - \gamma - \phi_a$$

$$\{\text{by Definition } 3.19\}$$

$$> r_{k,h} - G$$

Subcase 2: $T_{a,b}$ *is ready at time* t_p^-. In this case, because $T_{k,h}$ is scheduled at t_p^- and $T_{a,b}$ is not scheduled, we have

$$r_{k,h} - G = \chi(T_{k,h}, t_p^-) \leq \chi(T_{a,b}, t_p^-) = \chi^A(T_{a,b}, t_p^-). \tag{A.27}$$

The latter equality holds because $T_{a,b}$ is not scheduled at t_p^- and thus is not executing non-preemptively then. By (A.23) and (A.25), $d_{a,b} + M < t_p^- < t_p$. Therefore, by Definition 3.18, we have

$$\chi^A(T_{a,b}, t_p^-) \leq \chi^A(T_{a,b}, t_p). \tag{A.28}$$

By (A.27) and (A.28), we have $r_{k,h} - G \leq \chi^A(T_{a,b}, t_p)$. If $r_{k,h} - G < \chi^A(T_{a,b}, t_p)$ holds, then $T_{a,b}$ cannot preempt $T_{k,h}$. If $r_{k,h} - G = \chi^A(T_{a,b}, t_p)$, then by (A.27) and (A.28), we have $r_{k,h} - G = \chi^A(T_{a,b}, t_p^-)$, and hence, the tie-breaking between jobs $T_{a,b}$ and $T_{k,h}$ is not consistent. $\square$

Appendix B

Proofs for Lemmas in Chapter 5

In this appendix, we prove Claim 5.6, Lemmas 5.6, 5.7, 5.8, 5.18, and 5.19. We first prove Claim 5.6.

Claim 5.6: $\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1}) \leq r_{\ell,q} + \Theta_\ell - r_{\ell,q-\lambda+1}$.

Proof. Each job $T_{\ell,q-k}$, where $k \geq \lambda$ completes by $f_{\ell,q-\lambda}$. Thus,

$$\sum_{k \geq \lambda} W(T_{\ell,q-k}, r_{\ell,q-\lambda+1})$$

$$\{\text{by Definition 5.19}\}$$

$$\leq f_{\ell,q-\lambda} - r_{\ell,q-\lambda+1}$$

$$\{\text{by Definition 5.11}\}$$

$$\leq r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda) - r_{\ell,q-\lambda+1}. \tag{B.1}$$

Also, by Definition 5.19,

$$\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1})$$

$$= \sum_{T_{\ell,j} \in \mathcal{J}} W(T_{\ell,j}, r_{\ell,q-\lambda+1})$$

$$\{\text{because } \mathcal{J} \text{ does not contain } T_{\ell,q}\text{'s successors}\}$$

$$= \sum_{k \geq 0} W(T_{\ell,q-k}, r_{\ell,q-\lambda+1})$$

$$= \sum_{k \geq \lambda} W(T_{\ell,q-k}, r_{\ell,q-\lambda+1}) + \sum_{k \in [0,\lambda-1]} W(T_{\ell,q-k}, r_{\ell,q-\lambda+1})$$

$$\{\text{by (B.1)}\}$$

$$\leq r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda) - r_{\ell,q-\lambda+1} + \sum_{k \in [0,\lambda-1]} W(T_{\ell,q-k}, r_{\ell,q-\lambda+1})$$

$$\{\text{by Definitions 5.1 and 5.19}\}$$

$$\leq r_{\ell,q} + \Theta_\ell - \gamma_\ell^u(\lambda) - r_{\ell,q-\lambda+1} + \gamma_\ell^u(\lambda)$$

$$= r_{\ell,q} + \Theta_\ell - r_{\ell,q-\lambda+1}. \qquad \square$$

Because the allocation of a task over a set of intervals cannot exceed the cumulative length of these intervals, the claim below follows.

Claim B.1: $\mathsf{A}_{\mathcal{J}}(T_i, [t_0(k), r_{\ell,q} + \Theta_\ell)) \leq r_{\ell,q} - t_0(k) + \Theta_\ell.$

Lemma 5.6: $\mathsf{A_{NC}}(T_i, \delta) = \min(\delta + \Theta_\ell, \gamma_i^u(\alpha_i^+(\delta + C_{\ell,i}))).$

Proof. The competing demand due to T_i is upper-bounded by the demand due to T_i's jobs in $\mathcal{J}$ (refer to Definition 5.14). Because $T_i \in \mathbf{NC}$, all such jobs released prior to $t_0(k)$ are completed by time $t_0(k)$. For any $T_{i,j} \in \mathcal{J}$, by Lemma 5.5, $r_{i,j} \leq r_{\ell,q} + C_{\ell,i}$. Therefore, the allocation $\mathsf{A}_{\mathcal{J}}(T_i, [t_0(k), r_{\ell,q} + \Theta_\ell))$ is upper-bounded by the total execution time of T_i's jobs released within $[t_0(k), r_{\ell,q} + C_{\ell,i}]$. From Definitions 5.1 and 5.10, we have

$$\mathsf{A}_{\mathcal{J}}(T_i, [t_0(k), r_{\ell,q} + \Theta_\ell)) \leq \gamma_i^u(\alpha_i^+(r_{\ell,q} + C_{\ell,i} - t_0(k)))$$

$$= \gamma_i^u(\alpha_i^+(r_{\ell,q} - t_0(k) + C_{\ell,i})).$$

By Claim B.1 and the inequality above, $\mathsf{A_{NC}}(T_i, \delta)$ upper-bounds $\mathsf{A}_{\mathcal{J}}(T_i, [t_0(k), r_{\ell,q} + \Theta_\ell))$ for $\delta = r_{\ell,q} - t_0(k)$. $\qquad \square$

The following claim and a lemma will be used to prove Lemmas 5.7 and 5.9.

Claim B.2. *The function $G_i(S, X)$ as defined in Definition 5.26 is a non-decreasing function of the integral argument S.*

Proof. Suppose that $S \geq 1$ is fixed. We compute $G_i(S + 1, X)$. By Definition 5.26,

$$G_i(S + 1, X) = \min(\gamma_i^u(S + 1), \max(0, X - \mathcal{A}_\ell^{-1}(S)) + \gamma_i^u(S))$$

$$\{\text{because } \gamma_i^u(S) \text{ is a non-decreasing function}\}$$

$$\geq \gamma_i^u(S)$$

$$\geq \min(\gamma_i^u(S), \max(0, X - \mathcal{A}_\ell^{-1}(S - 1)) + \gamma_i^u(S - 1))$$

$$= G_i(S, X). \qquad \square$$

Lemma B.1. *If $t_x \leq r_{\ell,q}$, then*

$$\mathsf{W}_{\mathcal{J}}(T_i, t_x) \leq G_i(\alpha_i^u(r_{\ell,q} - t_x + C_{\ell,i} + \Theta_i), r_{\ell,q} - t_x + C_{\ell,i} + \Theta_i)$$

Proof. Let $T_{i,c} \in \mathcal{J}$ be the earliest job of T_i that is pending at or after time t_x. Note that if $T_{i,c}$ does not exit, then $\mathsf{W}_{\mathcal{J}}(T_i, t_x) = 0$. From the selection of $T_{i,c}$, we have $f_{i,c} > t_x$. If $T_{i,c} \neq T_{\ell,q}$, then $T_{i,c} \prec T_{\ell,q}$, which, by (5.12), implies

$$f_{i,c} > t_x \wedge r_{i,c} + \Theta_i > t_x. \tag{B.2}$$

If $T_{i,c} = T_{\ell,q}$, then

$$f_{i,c} > t_x \wedge r_{i,c} + \Theta_i - t_x \geq \gamma_i^u(1). \tag{B.3}$$

The predicate above holds because $t_x \leq r_{\ell,q}$ by the condition of the lemma, and $\Theta_i \geq \gamma_i^u(1) > 0$ by Claim 5.1. Note that (B.3) implies (B.2). We define the job set $\mathcal{J}_i$ as follows.

$$\text{Let } \mathcal{J}_i = \{T_{i,y} : y \geq c \wedge T_{i,y} \in \mathcal{J}\}. \tag{B.4}$$

To establish an upper-bound on $\mathsf{W}_{\mathcal{J}}(T_i, t_x)$, we first rewrite $\mathsf{W}_{\mathcal{J}}(T_i, t_x)$ as follows.

$$\mathsf{W}_{\mathcal{J}}(T_i, t_x) = W(T_{i,c}, t_x) + \sum_{T_{i,y} \in \mathcal{J} \setminus T_{i,c}} W(T_{i,y}, t_x)$$

$$= W(T_{i,c}, t_x) + \sum_{T_{i,y} \in \mathcal{J}_i \setminus T_{i,c}} W(T_{i,y}, t_x) \tag{B.5}$$

We now bound the $W(T_{i,c}, t_x)$ term in (B.5) by considering two cases.

Case 1: $T_{i,c} = T_{\ell,q}$. By (B.3), $r_{\ell,q} + \Theta_\ell - t_x \geq \gamma_\ell^u(1) \geq e_{\ell,q}$, in which case $W(T_{\ell,q}, t_x) \leq e_{\ell,q}$. Thus,

$$W(T_{\ell,q}, t_x) \leq \min(e_{\ell,q}, r_{\ell,q} + \Theta_\ell - t_x). \tag{B.6}$$

Case 2: $T_{i,c} \neq T_{\ell,q}$. By (B.2), $T_{i,c}$ finishes its execution at time $f_{i,c} > t_x$, and hence,

$$W(T_{i,c}, t_x) \leq \min(e_{i,c}, f_{i,c} - t_x)$$

$$\{\text{if } T_{i,c} \prec T_{\ell,q}, \text{ by (5.12)}\}$$

165

$$\leq \min(e_{i,c}, r_{i,c} + \Theta_i - t_x). \tag{B.7}$$

By (B.5), (B.6), and (B.7),

$$
\begin{aligned}
\mathsf{W}_{\mathcal{J}}(T_i, t_x) & \\
\leq\ & \min(e_{i,c}, r_{i,c} + \Theta_i - t_x) + \sum_{T_{i,y} \in \mathcal{J}_i \setminus T_{i,c}} W(T_{i,y}, t_x) \\
\leq\ & \min\left(e_{i,c} + \sum_{T_{i,y} \in \mathcal{J}_i \setminus T_{i,c}} W(T_{i,y}, t_x), r_{i,c} + \Theta_i - t_x + \sum_{T_{i,y} \in \mathcal{J}_i \setminus T_{i,c}} W(T_{i,y}, t_x) \right).
\end{aligned}
\tag{B.8}
$$

Let

$$S_i = |\mathcal{J}_i|. \tag{B.9}$$

Because the execution demand of job $T_{i,y}$ cannot be greater than its execution time, by Definition 5.1, we have the following.

$$e_{i,c} + \sum_{T_{i,y} \in \mathcal{J}_i \setminus T_{i,c}} W(T_{i,y}, t_x) \leq \gamma_i^u(S_i) \tag{B.10}$$

$$\sum_{T_{i,y} \in \mathcal{J}_i \setminus T_{i,c}} W(T_{i,y}, t_x) \leq \gamma_i^u(S_i - 1) \tag{B.11}$$

By (B.8), (B.10), and (B.11), we have

$$\mathsf{W}_{\mathcal{J}}(T_i, t_x) \leq \min(\gamma_i^u(S_i), r_{i,c} + \Theta_i - t_x + \gamma_i^u(S_i - 1)). \tag{B.12}$$

We next establish an upper bound on $r_{i,c}$ in (B.12). From (B.4) above and Lemma 5.5, we have

(**R**) If $T_{i,y} \in \mathcal{J}_i$, then $r_{i,y} \in [r_{i,c}, r_{\ell,q} + C_{\ell,i}]$.

By (B.9), $T_{i,c+S_i-1}$ is the latest job of T_i released within $[r_{i,c}, r_{\ell,q} + C_{\ell,i}]$. We upper bound $r_{i,c}$ as follows.

$$r_{i,c} = r_{i,c+S_i-1} + r_{i,c} - r_{i,c+S_i-1}$$

$$\{\text{by the definition of } T_{i,c+S_i-1}\}$$

$$\leq r_{\ell,q} + C_{\ell,i} + r_{i,c} - r_{i,c+S_i-1}$$

$$\{\text{by Lemma 5.2}\}$$

$$\leq r_{\ell,q} + C_{\ell,i} - \mathcal{A}_i^{-1}(S_i - 1)$$

From the inequality above, we have

$$
\begin{aligned}
r_{i,c} + \Theta_i - t_x &\leq \max(0, r_{\ell,q} + C_{\ell,i} - \mathcal{A}_i^{-1}(S_i-1) + \Theta_i - t_x) \\
&= \max(0, r_{\ell,q} - t_x + C_{\ell,i} + \Theta_i - \mathcal{A}_i^{-1}(S_i - 1)).
\end{aligned}
\tag{B.13}
$$

By (B.12) and (B.13), we have

$$
\begin{aligned}
\mathsf{W}_{\mathcal{J}}&(T_i, t_x) \\
&\leq \min(\gamma_i^u(S_i), \max(0, r_{\ell,q} - t_x + C_{\ell,i} + \Theta_i - \mathcal{A}^{-1}(S_i - 1)) + \gamma_i^u(S_i - 1)) \\
&= G_i(S_i, r_{\ell,q} - t_x + C_{\ell,i} + \Theta_i),
\end{aligned}
\tag{B.14}
$$

where $G_i(S, X)$ is defined as in Definition 5.26. By Claim B.2, the function $G_i(S, X)$ is a non-decreasing function of S. We thus can find an upper bound on $\mathsf{W}_{\mathcal{J}}(T_i, t_x)$ by setting an upper bound on S_i into (B.14).

By (R), $S_i = |\mathcal{J}_i|$ is at most the number of jobs of T_i released within the interval $[r_{i,c}, r_{\ell,q} + C_{\ell,i}]$, which, by (B.2), is contained within $(t_x - \Theta_i, r_{\ell,q} + C_{\ell,i}]$. We thus upper bound S_i using Definition 5.2.

$$S_i \leq \alpha_i^u(r_{\ell,q} - t_x + C_{\ell,i} + \Theta_i)$$

Setting this upper bound on S_i into (B.14), we get the conclusion of the lemma. $\qquad\square$

Using the result of the lemma above, we next prove Lemma 5.7.

Lemma 5.7: $\mathsf{A_{HC}}(T_i, \delta) = \min(\delta + \Theta_\ell, G_i(\alpha_i^u(\delta + C_{\ell,i} + \Theta_i), \delta + C_{\ell,i} + \Theta_i))$.

Proof. Consider $T_i \in \mathbf{HC}$. The allocation of T_i's jobs from $\mathcal{J}$ cannot exceed their cumulative demand.

From Definitions 5.15 and 5.19, we have

$$\mathsf{A}_{\mathcal{J}}(T_i, [t_0(k), r_{\ell,q} + \Theta_\ell)) \leq \mathsf{W}_{\mathcal{J}}(T_i, t_0(k))$$

$$\{\text{by Lemma B.1}\}$$

$$\leq G_i(\alpha_i^u(t_0(k) - r_{\ell,q} + C_{\ell,i} + \Theta_i), t_0(k) - r_{\ell,q} + C_{\ell,i} + \Theta_i)$$

$$\{\text{setting } \delta = t_0(k) - r_{\ell,q}\}$$

$$= G_i(\alpha_i^u(\delta + C_{\ell,i} + \Theta_i), \delta + C_{\ell,i} + \Theta_i).$$

By the inequality above and Claim B.1, $\mathsf{A}_{\mathbf{HC}}(T_i, \delta)$ upper-bounds $\mathsf{A}_{\mathcal{J}}(T_i, [t_0(k), r_{\ell,q} + \Theta_\ell))$ for $\delta = r_{\ell,q} - t_0(k)$. $\qquad\square$

The following claims and lemma are used to prove Lemma 5.8.

Claim B.3: $L_i(X + Y) \leq L_i(X) + u_i \cdot Y$ for all X and $Y \geq 0$.

Proof. By Definition 5.4, $u_i > 0$. By the condition of the claim, $Y \geq 0$. Thus, by Definition 5.27,

$$L_i(X + Y) = \max(0, u_i \cdot (X + Y) + \overline{e_i} \cdot B_i) + v_i$$

$$\leq \max(0, u_i \cdot X + \overline{e_i} \cdot B_i) + v_i + u_i \cdot Y$$

$$= L_i(X) + u_i \cdot Y. \qquad\square$$

Claim B.4. $\alpha_i^+(X) \leq R_i \cdot X + B_i$ *for* $X \geq 0$.

Proof. By Definition 5.10,

$$\alpha_i^+(X) = \lim_{\epsilon \to +0} \alpha_i^u(X + \epsilon)$$

$$\{\text{by (5.2)}\}$$

$$\leq \lim_{\epsilon \to +0} R_i \cdot (X + \epsilon) + B_i$$

$$= R_i \cdot X + B_i. \qquad\square$$

Claim B.5: $\gamma_i^u(\alpha_i^u(X)) \leq \gamma_i^u(\alpha_i^+(X)) \leq L_i(X)$ for all X.

Proof. By Definition 5.2, $\alpha_i^u(\Delta)$ is a non-decreasing function of Δ. Therefore, $\alpha_i^u(\Delta) \leq \alpha_i^u(\Delta + \epsilon)$ for any $\epsilon > 0$, which implies $\alpha_i^u(\Delta) \leq \lim_{\epsilon \to +0} \alpha_i^u(\Delta + \epsilon)$. The right-hand side of the latter inequality is

$\alpha_i^+(\Delta)$ by Definition 5.10. Thus, $\alpha_i^u(\Delta) \leq \alpha_i^+(\Delta)$. The first inequality of the claim therefore follows from $\gamma_i^u(k)$ being a non-decreasing function of k by Definition 5.1. We now prove the second inequality by considering two cases.

Case 1: $X < 0$. In this case, by Definition 5.10, $\alpha_i^+(X) = 0$. By Definition 5.1, $\gamma_i^u(\alpha_i^+(X)) = 0$. The required result follows from Definition 5.27 and $v_i \geq 0$ (see (5.3)).

Case 2: $X \geq 0$. By Definition 5.10, because $\alpha_i^+(X) \geq 0$, we have

$$\gamma_i^u(\alpha_i^+(X)) = \gamma_i^u(\max(0, \alpha_i^+(X)))$$

$$\{\text{by (5.3)}\}$$

$$\leq \overline{e_i} \cdot (\max(0, \alpha_i^+(X))) + v_i$$

$$\{\text{by Claim B.4}\}$$

$$\leq \overline{e_i} \cdot (\max(0, R_i \cdot X + B_i)) + v_i$$

$$= \max(0, \overline{e_i} \cdot R_i \cdot X + \overline{e_i} \cdot B_i) + v_i$$

$$\{\text{by Definition 5.4}\}$$

$$= \max(0, u_i \cdot X + \overline{e_i} \cdot B_i) + v_i$$

$$\{\text{by Definition 5.27}\}$$

$$= L_i(X). \qquad \square$$

Lemma B.2: $\mathsf{A_{HC}}(T_i, \delta) \leq L_i(\delta + C_{\ell,i}) + u_i \cdot \Theta_i$ and $\mathsf{A_{NC}}(T_i, \delta) \leq L_i(\delta + C_{\ell,i})$.

Proof. We prove the first inequality.

$$\mathsf{A_{HC}}(T_i, \delta)$$

$$\{\text{by Lemma 5.7}\}$$

$$= \min(\delta + \Theta_\ell, G_i(\alpha_i^u(\delta + C_{\ell,i} + \Theta_i), \delta + C_{\ell,i} + \Theta_i))$$

$$\leq G_i(\alpha_i^u(\delta + C_{\ell,i} + \Theta_i), \delta + C_{\ell,i} + \Theta_i)$$

$$\{\text{by Definition 5.26}\}$$

$$\leq \gamma_i^u(\alpha_i^u(\delta + C_{\ell,i} + \Theta_i))$$

$$\{\text{by Claim B.5}\}$$

$$\leq L_i(\delta + C_{\ell,i} + \Theta_i)$$

$$\{\text{because } \Theta_i \geq 0, \text{ by Claim B.3}\}$$

$$\leq L_i(\delta + C_{\ell,i}) + u_i \cdot \Theta_i$$

The second inequality is proved similarly.

$$\mathsf{A_{NC}}(T_i, \delta)$$

$$\{\text{by Lemma 5.6}\}$$

$$= \min(\delta + \Theta_\ell, \gamma_i^u(\alpha_i^+(\delta + C_{\ell,i})))$$

$$\leq \gamma_i^u(\alpha_i^+(\delta + C_{\ell,i}))$$

$$\{\text{by Claim B.5}\}$$

$$\leq L_i(\delta + C_{\ell,i}) \qquad \square$$

Lemma 5.8. *For all $\delta \geq 0$, $M_\ell^*(\delta) \leq U_{sum} \cdot \delta + H_\ell$, where $H_\ell = \sum_{T_i \in \tau} L_i(C_{\ell,i}) + U(m-1) \cdot \max(\Theta_i)$ and $U(y)$ is the sum of $\min(y, |\tau|)$ largest utilizations.*

Proof. Suppose that the sets **HC** and **NC** subject to (5.28) maximize the value of the right-hand side of (5.27). By (5.27), we have

$$M_\ell^*(\delta) = \sum_{T_i \in \mathbf{HC}} \mathsf{A_{HC}}(T_i, \delta) + \sum_{T_i \in \mathbf{NC}} \mathsf{A_{NC}}(T_i, \delta)$$

$$\{\text{by Lemma B.2}\}$$

$$\leq \sum_{T_i \in \mathbf{HC}} (L_i(\delta + C_{\ell,i}) + u_i \cdot \Theta_i) + \sum_{T_i \in \mathbf{NC}} L_i(\delta + C_{\ell,i})$$

$$\{\text{since } \mathbf{HC} \cup \mathbf{NC} \subseteq \tau \text{ and } L_i(X) \geq 0 \text{ for all } X\}$$

$$\leq \sum_{T_i \in \tau} L_i(\delta + C_{\ell,i}) + \sum_{T_i \in \mathbf{HC}} u_i \cdot \Theta_i$$

$$\left\{ \begin{array}{l} \text{because } |\mathbf{HC}| \leq m-1 \text{ by (5.28), and using the definition} \\ \text{of } U(y) \text{ in the statement of the lemma} \end{array} \right\}$$

$$\leq \sum_{T_i \in \tau} [L_i(\delta + C_{\ell,i})] + U(m-1) \cdot \max(\Theta_i)$$

$$\{\text{by Claim B.3 (note that, by the statement of the lemma, } \delta \geq 0)\}$$

$$\leq \sum_{T_i \in \tau} [L_i(C_{\ell,i}) + u_i \cdot \delta] + U(m-1) \cdot \max(\Theta_i)$$

$$\left\{ \begin{array}{l} \text{by Definition 5.4 and the definition of } H_\ell \\ \text{in the statement of the lemma} \end{array} \right\}$$

$$= U_{sum} \cdot \delta + H_\ell. \qquad \qquad \square$$

We now derive a lower bound for $E_\ell^*(\lambda)$ given by Lemma 5.9 and prove Lemma 5.18.

Lemma 5.9. *If $E_\ell^*(k)$ is given by (5.29), then $E_\ell^*(\lambda) \geq \mathsf{W}_{\mathcal{J}}(T_i, r_{\ell,q-\lambda+1})$.*

Proof. By Definition 5.20, the function $E_\ell^*(\lambda)$ upper bounds $\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1})$, which is the amount of work due to unfinished jobs of T_ℓ in $\mathcal{J}$ at time $r_{\ell,q-\lambda+1}$. By Lemma B.1,

$$\mathsf{W}_{\mathcal{J}}(T_\ell, r_{\ell,q-\lambda+1})$$

$$\leq G_\ell(\alpha_\ell^u(r_{\ell,q} - r_{\ell,q-\lambda+1} + C_{\ell,\ell} + \Theta_\ell), r_{\ell,q} - r_{\ell,q-\lambda+1} + C_{\ell,\ell} + \Theta_\ell)$$

$$\{\text{because } C_{\ell,\ell} = 0 \text{ by Definition 5.24}\}$$

$$= G_\ell(\alpha_\ell^u(r_{\ell,q} - r_{\ell,q-\lambda+1} + \Theta_\ell), r_{\ell,q} - r_{\ell,q-\lambda+1} + \Theta_\ell)$$

$$\{\text{by Claim 5.9}\}$$

$$\leq G_\ell(\alpha_\ell^u(\max(0, \gamma_\ell^u(\lambda - 1) - 1) + \Theta_\ell), \max(0, \gamma_\ell^u(\lambda - 1) - 1) + \Theta_\ell)$$

$$\{\text{by (5.29)}\}$$

$$= E_\ell^*(\lambda). \qquad \qquad \square$$

Lemma 5.18. *If $\Theta_\ell = x + \gamma_\ell^u(K_\ell) + C_\ell$, where $x \geq 0$, then $E_\ell^*(k) \leq Y_\ell + u_\ell \cdot x$ for $k \in [1, K_\ell]$.*

Proof. By (5.29),

$$E_\ell^*(k) = G_\ell(\alpha_\ell^u(Q(k)), Q(k))$$

$$\{\text{by Definition 5.26}\}$$

$$\leq \gamma_\ell^u(\alpha_\ell^u(Q(k)))$$

$$\{\text{by Claim B.5}\}$$

$$\leq L_\ell(Q(k))$$

$$\{\text{by Definition 5.28}\}$$

$$= L_\ell\big(\max(0, \gamma_\ell^u(k - 1) - 1) + \Theta_\ell\big)$$

171

{by the condition of the Lemma}

$$= L_\ell\big(\max(0, \gamma_\ell^u(k-1) - 1) + x + \gamma_\ell^u(K_\ell) + C_\ell\big)$$

{by Claim B.3}

$$\leq L_\ell\big(\max(0, \gamma_\ell^u(k-1) - 1) + \gamma_\ell^u(K_\ell) + C_\ell\big) + u_\ell \cdot x$$

$$\left\{ \begin{array}{l} \text{because } L_\ell \text{ and } \gamma_\ell^u \text{ are non-decreasing} \\ \text{functions of their arguments} \end{array} \right\}$$

$$\leq L_\ell\big(\max(0, \gamma_\ell^u(K_\ell-1) - 1) + \gamma_\ell^u(K_\ell) + C_\ell\big) + u_\ell \cdot x$$

{by Definition 5.35}

$$= Y_\ell + u_\ell \cdot x. \qquad \square$$

Lemma 5.19. *If* $\Theta_i = x + \gamma_i^u(K_i) + C_i$ *for each task* T_i *and* $\delta \geq 0$, *then* $M_\ell^*(\delta) \leq U_{sum} \cdot \delta + U(m-1) \cdot x + \mathcal{W} + \sum_{T_i \in \tau} L_i(C_{\ell,i})$, *where* $U(m-1)$ *is the sum of* $m-1$ *largest task utilizations.*

Proof. Suppose that the sets **HC** and **NC** subject to (5.28) maximize the value of the right-hand side of (5.27). By (5.27), we have

$$M_\ell^*(\delta)$$

$$= \sum_{T_i \in \mathbf{HC}} \mathsf{A}_{\mathbf{HC}}(T_i, \delta) + \sum_{T_i \in \mathbf{NC}} \mathsf{A}_{\mathbf{NC}}(T_i, \delta)$$

{by Lemma B.2}

$$\leq \sum_{T_i \in \mathbf{HC}} (L_i(\delta + C_{\ell,i}) + u_i \cdot \Theta_i) + \sum_{T_i \in \mathbf{NC}} L_i(\delta + C_{\ell,i})$$

{since $\mathbf{HC} \cup \mathbf{NC} \subseteq \tau$ and $L_i(X) \geq 0$ for all X}

$$\leq \sum_{T_i \in \tau} L_i(\delta + C_{\ell,i}) + \sum_{T_i \in \mathbf{HC}} u_i \cdot \Theta_i$$

{by the selection of Θ_i in the statement of the Lemma}

$$= \sum_{T_i \in \tau} L_i(\delta + C_{\ell,i}) + \sum_{T_i \in \mathbf{HC}} u_i \cdot (x + \gamma_i^u(K_i) + C_i)$$

$$\left\{ \begin{array}{l} \text{because } |\mathbf{HC}| \leq m - 1 \text{ by (5.28), and using the} \\ \text{definition of } U(y) \text{ in the statement of the lemma} \end{array} \right\}$$

$$\leq \sum_{T_i \in \tau} L_i(\delta + C_{\ell,i}) + U(m-1) \cdot x + \sum_{T_i \in \mathbf{HC}} u_i \cdot (\gamma_i^u(K_i) + C_i)$$

{because $|\mathbf{HC}| \leq m - 1$ by (5.28), and by Definition 5.36}

$$\leq \sum_{T_i \in \tau} L_i(\delta + C_{\ell,i}) + U(m-1) \cdot x + \mathcal{W}$$

$$\left\{ \begin{array}{l} \text{by Claim B.3} \\[4pt] \text{(note that, by the condition of the lemma, } \delta \geq 0) \end{array} \right\}$$

$$\leq \sum_{T_i \in \tau} [L_i(C_{\ell,i}) + u_i \cdot \delta] + U(m-1) \cdot x + \mathcal{W}$$

{by Definition 5.4}

$$= U_{sum} \cdot \delta + \sum_{T_i \in \tau} L_i(C_{\ell,i}) + U(m-1) \cdot x + \mathcal{W}. \qquad \square$$

BIBLIOGRAPHY

(2007). Linux vserver documentation. http://linux-vserver.org/Documentation.

(2009). http://rt.wiki.kernel.org/index.php/Main_Page.

Anderson, J., Calandrino, J., and Devi, U. (2006). Real-time scheduling on multicore platforms. In *Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 179–190.

Anderson, J. and Srinivasan, A. (2000). Pfair scheduling: Beyond periodic task systems. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications*, pages 297–306.

Anderson, J. and Srinivasan, A. (2004). Mixed Pfair/ERfair scheduling of asynchronous periodic tasks. *Journal of Computer and System Sciences*, 68(1):157–204.

Atlas, A. and Bestavros, A. (1998). Statistical rate monotonic scheduling. In *Proceedings of the 19th Real-Time Systems Symposium*, pages 123–132.

Baker, T. P. (2003). Multiprocessor EDF and deadline monotonic schedulability analysis. In *Proceedings of the 24th IEEE Real-Time Systems Symposium*, pages 120–129.

Baruah, S. (2003). Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors. *IEEE Transactions on Computers*, 53(6):781–784.

Baruah, S. (2007). Techniques for multiprocessor global schedulability analysis. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 119–128.

Baruah, S. and Baker, T. (2008). Global EDF schedulability analysis of arbitrary sporadic task systems. In *Proceedings of the EuroMicro Conference on Real-Time Systems*, pages 3–12.

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

Baruah, S. and Fisher, N. (2006). The partitioned multiprocessor scheduling of deadline-constrained sporadic task systems. *IEEE Transactions on Computers*, 55(7):918–923.

Baruah, S. and Fisher, N. (2007). The partitioned dynamic-priority scheduling of sporadic task systems. *Real-Time Systems*, 36(3):199–226.

Bennett, E. (2007). *Computational Video Enhancement*. PhD thesis, The University of North Carolina at Chapel Hill.

Bennett, E. and McMillan, L. (2005). Video enhancement using per-pixel virtual exposures. *ACM Transactions on Graphics (SIGGRAPH)*, 24(3):845–852.

Bertogna, M., Cirinei, M., and Lipari, G. (2008). Schedulability analysis of global scheduling algorithms on multiprocessor platforms. *IEEE Transactions on Parallel and Distributed Systems*, 20(4):553–566.

Bini, E., Bertogna, M., and Baruah, S. (2009a). Virtual multiprocessor platforms: Specification and use. In *Proceedings of the 30th Real-Time Systems Symposium*. To appear.

Bini, E., Buttazzo, G., and Bertogna, M. (2009b). The multi-supply function abstraction for multiprocessors. In *Proceedings of the 15th Embedded and Real-Time Computing Systems and Applications*, pages 294–302.

Block, A. (2008). *Adaptive Multiprocessor Real-Time Systems.* PhD thesis, The University of North Carolina at Chapel Hill.

Block, A., Brandenburg, B., Anderson, J., and Quint, S. (2008). An adaptive framework for multiprocessor real-time systems. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems*, pages 23–33.

Block, A., Leontyev, H., Brandenburg, B., and Anderson, J. (2007). A flexible real-time locking protocol for multiprocessors. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 71–80.

Blum, M., Floyd, R., Pratt, V., Rivest, R., and Tarjan, R. (1973). Time bounds for selection. *Journal of Computer and System Sciences*, 7(4):448–461.

Brandenburg, B. and Anderson, J. (2009). On the implementation of global real-time schedulers. In *Proceedings of the 30th IEEE Real-Time Systems Symposium*, pages 214–227.

Brandenburg, B., Block, A., Calandrino, J., Devi, U., Leontyev, H., and Anderson, J. (2007). LITMUS^RT: A status report. In *Proceedings of the 9th Real-Time Linux Workshop*, pages 107–123.

Brandenburg, B., Calandrino, J., and Anderson, J. (2008a). On the scalability of real-time scheduling algorithms on multicore platforms: A case study. In *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 157–169.

Brandenburg, B., Calandrino, J., Block, A., Leontyev, H., and Anderson, J. (2008b). Real-time synchronization on multiprocessors: To block or not to block, to suspend or spin? In *Proceedings of the 14th IEEE Real-Time and Embedded Technology and Applications Symposium*, page 342353.

Brandenburg, B., Leontyev, H., and Anderson, J. (2009). Accounting for interrupts in multiprocessor real-time systems. In *Proceedings of the 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 273–283.

Calandrino, J. (2009). *On the Design and Implementation of a Cache-Aware Soft Real-Time Scheduler for Multicore Platforms.* PhD thesis, University of North Carolina at Chapel Hill.

Calandrino, J., Anderson, J., and Baumberger, D. (2007). A hybrid rreal-time scheduling approach for largescale multicore platforms. In *Proceedings of Euromicro Conference on Real-Time Systems*, pages 247–258.

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 International Real-Time Systems Symposium*, pages 111–126.

Chakraborty, S., Kunzli, S., and Thiele, L. (2003). A general framework for analysing system properties in platform-based embedded system designs. In *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*.

Chakraborty, S., Liu, Y., Stoimenov, N., Thiele, L., and Wandeler, E. (2006). Interface-based rate analysis of embedded systems. In *Proceedings of the 27th IEEE Real-Time Systems Symp.*, pages 25–34.

Chakraborty, S. and Thiele, L. (2005). A new task model for streaming applications and its schedulability analysis. In *Proceedings of the IEEE Design Automation and Test in Europe (DATE)*, pages 486–491.

Chen, J., Stoimenov, N., and Thiele, L. (2009). Feasibility analysis of on-line DVS algorithms for scheduling arbitrary event streams. In *Proceedings of the 30th Real-Time Systems Symposium*.

Cho, H., Ravindran, B., and Jensen, E. (2006). An optimal real-time scheduling algorithm for multiprocessors. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*.

Cho, S., Lee, S., Ahn, S., and Lin, K. (2002). Efficient real-time scheduling algorithms for multiprocessor systems. *IEICE Transactions on Communications*, E85-B(12):2859–2867.

Chuprat, S. and Baruah, S. (2008). Scheduling divisible real-time loads on clusters with varying processor start times. In *Proceedings of the IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 15–24.

Cirinei, M. and Baker, T. (2007). EDZL scheduling analysis. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems*, pages 9–18.

Cruz, R. (1991a). A calculus for network delay, Part I: Network elements in isolation. *IEEE Transactions on Information Theory*, 37(1).

Cruz, R. (1991b). A calculus for network delay, Part II: Network analysis. *IEEE Transactions on Information Theory*, 37(1).

Cruz, R. (1995). Quality of service guarantees in virtual circuit switched networks. *IEEE Journal of Selected Areas in Communications*, 13(6):1048–1056.

Devi, U. (2006). *Soft Real-Time Scheduling on Multiprocessors*. PhD thesis, The University of North Carolina at Chapel Hill.

Devi, U. and Anderson, J. (2005). Tardiness bounds for global EDF scheduling on a multiprocessor. In *Proceedings of the 26th IEEE Real-Time Systems Symposium*, pages 330–341.

Devi, U. and Anderson, J. (2006). Flexible tardiness bounds for sporadic real-time task systems on multiprocessors. In *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium*. On CDROM.

Devi, U. and Anderson, J. (2008a). A schedulable utilization bound for the multiprocessor epdf pfair algorithm. *Real-Time Systems*, 38(3):237–288.

Devi, U. and Anderson, J. (2008b). Tardiness bounds under global edf scheduling on a multiprocessor. *Real-Time Systems*, 38(2):133–189.

Devi, U. and Anderson, J. (2009). Improved conditions for bounded tardiness under epdf pfair multiprocessor scheduling. *Journal of Computer and System Sciences*, 75(7):388–420.

Devi, U., Leontyev, H., and Anderson, J. (2006). Efficient synchronization under global EDF scheduling on multiprocessors. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, pages 75–84.

Easwaran, A., Shin, I., and Lee, I. (2009). Optimal virtual cluster-based multiprocessor scheduling. *Real-Time Systems*, 43(1):25–59.

Eriksson, M. and Palmroos, S. (2007). Comparative study of containment strategies in solaris and security enhanced Linux. http://opensolaris.org/os/community/security/news/20070601-thesis-bs-eriksson-palmroos.pdf.

Golab, L., Johnson, T., Spencer, J., and Shkapenyuk, V. (2009). Stream warehousing with DataDepot. In *Proceedings of SIGMOD*, pages 847–854.

Goossens, J., Funk, S., and Baruah, S. (2003). Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Systems*, 25(2-3):187–205.

Guan, N., Stigge, M., Yi, W., and Yu, G. (2009). New response-time bounds for fixed priority multiprocessor scheduling. In *Proceedings of the 30th Real-Time Systems Symposium*, pages 387–397.

Guan, N., Yi, W., Gu, Z., Deng, Q., and Yu, G. (2008). New schedulability test conditions for non-preemptive scheduling on multiprocessor platforms. In *Proceedings of Real-Time Systems Symposium*, pages 137–146.

Hamdaoui, M. and Ramanathan, P. (1995). A dynamic priority assignment technique for streams with (m,k)-firm guarantees. *IEEE Transactions on Computers*, 44(12):1443–1451.

Huang, K., Thiele, L., Stefanov, T., and Deprettere, E. (2007). Performance analysis of multimedia applications using correlated streams. In *Design, Automation and Test in Europe (DATE 07)*, pages 912–917.

Intel Corporation (2006). Intel digital home software vision guide 2007. http://isdlibrary.inteldispatch.com/isd/42/SSPR DigHomeGuide 2007.pdf.

LeBoudec, J. Y. and Thiran, P. (2001). *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet.* Number 2050. Springer Verlag.

Leontyev, H. and Anderson, J. (2007a). Tardiness bounds for EDF scheduling on multi-speed multicore platforms. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 103–111.

Leontyev, H. and Anderson, J. (2007b). Tardiness bounds for EDF scheduling on multi-speed multicore platforms. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 103–111.

Leontyev, H. and Anderson, J. (2007c). Tardiness bounds for FIFO scheduling on multiprocessors. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems.* 71-80.

Leontyev, H. and Anderson, J. (2008a). A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In *Proceedings of the 20th Euromicro Conference on Real-Time Systems*, pages 60–92.

Leontyev, H. and Anderson, J. (2008b). A unified hard/soft real-time schedulability test for global EDF multiprocessor scheduling. In *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 375–384.

Leontyev, H. and Anderson, J. (2009). A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. *Real-Time Systems*, 43(1):191–200.

Leontyev, H. and Anderson, J. (2010). Generalized tardiness bounds for global multiprocessor scheduling. *Real-Time Systems*, 44(1):26–71.

Lessard, P. (2003). Linux process containment: A practical look at chroot and user mode Linux. http://www.sans.org/reading_room/whitepapers/linux/1073.php.

Liu, C. and Layland, J. (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 30:46–61.

Liu, J. (2000). *Real-Time Systems.* Prentice Hall.

Lopez, J., Diaz, J., and Garcia, D. (2004). Utilization bounds for EDF scheduling on real-time multiprocessor systems. *Real-Time Systems*, 28(1):39–68.

Maxiaguine, A. (2005). *Modeling Multimedia Workloads for Embedded System Design.* PhD thesis, ETH Zurich.

Mok, A. K. and Chen, D. (1997). A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23:635–645.

Mok, A. K., Feng, X., and Chen, D. (2001). Resource partition for real-time systems. In *Proceedings of 7th Real-Time Technology and Applications Symposium*, pages 75–84.

Phan, L., Chakraborty, S., and Thiagarajan, P. (2008). A multi-mode real-time calculus. In *Proceedings of the 29th IEEE Real-Time Systems Symposium*, pages 59–69.

Piao, X., Han, S., Kim, H., Park, M., Cho, Y., and Cho, S. (2006). Predictability of earliest deadline zero laxity algorithm for multiprocessor real-time systems. In *Proceedings of the 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pages 359–364.

Rajkumar, R. (2006). Resource Kernels: Why Resource Reservation should be the Preferred Paradigm of Construction of Embedded Real-Time Systems. Keynote talk, 18th Euromicro Conference on Real-Time Systems, Dresden, Germany.

Richter, K., Jersak, M., and Ernst, R. (2003). A formal approach to MpSoC performance verification. *IEEE Computer*, 36(4):60–67.

Sariowan, H., Cruz, R., and Polyzos, G. (1995). Scheduling for quality-of-service guarantees via service curves. In *Proceedings of the International Conference on Computer Communications and Networks*, pages 512–520.

Stoica, I., Abdel-Wahab, H., Jeffay, K., Baruah, S., Gehrke, J., and Plaxton, C. (1996). A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 288–299.

Thiele, L. and Stoimenov, N. (2009). Modular performance analysis of cyclic dataflow graphs. In *EMSOFT 09: Proceedings of the 9th ACM international conference on Embedded software*, pages 127–136.

Vallidis, N. (2002). *WHISPER: A Spread Spectrum Approach to Occlusion in Acoustic Tracking*. PhD thesis, The University of North Carolina at Chapel Hill, North Carolina.

Wandeler, E. (2006). *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. PhD thesis, ETH Zurich.

Wandeler, E. and Thiele, L. (2006). Real-Time Calculus (RTC) Toolbox.

Wei, H., Chao, Y., Lin, S., Lin, K., and Shih, W. (2007). Current results on EDZL scheduling for multiprocessor real-time systems. In *Proceedings 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 120–130.

Wu, J., Liu, J.-C., and Zhao, W. (2005). On schedulability bounds of static priority schedulers. In *Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium*, pages 529–540.

Zhang, F. and Burns, A. (2009). Schedulability analysis for real-time systems with EDF scheduling. *IEEE Transactions on Computers*, 58(9):1250–1258.